NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DEVELOPMENT OF A TARGET RECOGNITION SYSTEM USING FORMAL AND SEMI-FORMAL SOFTWARE MODELING METHODS

by

Matthew A. Lisowski

December 2000

Thesis Co-Advisors: Neil Rowe
Man-Tak Shing

Approved for public release; distribution is unlimited

DTIC QUALITY INSPECTED 1

20010221 011

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY	2. REPORT DATE December 2000	3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE Development of a Target Recognition System Using Formal and Semi-Formal Software Modeling Methods		5. FUNDING NUMBERS
6. AUTHOR(S) Matthew A. Lisowski		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.		
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

With the shrinking defense budget, the U.S. Department of Defense (DoD) has relied more on commercial-off-the-shelf (COTS) and contracted software systems. Government contractors and commercial developers currently rely heavily on semi-formal methods such as the Unified Modeling Language (UML) in developing the models and requirements for these software systems. The correctness of specifications in such languages cannot be tested, in general, until they are implemented. Due to the inherent safety requirements for mission critical systems, formal specification methods would be preferable. This thesis contrasts the development of a combat system for the Navy using the formal specification language SPEC with development using the semi-formal method UML. The application being developed is a ship recognition system that utilizes image data, detected emitters, and ship positioning to correlate ship identification. The requirements analysis and architectural design for this system are presented.

14. SUBJECT TERMS: modeling, requirements analysis, formal specifications, uml, formal methods, semi-formal methods			15. NUMBER OF Pages 102
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

THIS PAGE IS INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

DEVELOPMENT OF A TARGET RECOGNITION SYSTEM USING FORMAL AND SEMI-FORMAL SOFTWARE MODELING METHODS

Matthew A. Lisowski
Lieutenant, United States Navy
B.S., United States Military Academy, 1991

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SOFTWARE ENGINERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2000

Author: [illegible]
Matthew A. Lisowski

Approved by: [illegible]
Neil Rowe, Thesis Co-Advisor

[illegible]
Man-Tak Shing, Thesis Co-Advisor

[illegible]
Luqi, Chair,
Software Engineering

THIS PAGE IS INTENTIONALLY LEFT BLANK

ABSTRACT

With the shrinking defense budget, the U.S. Department of Defense (DoD) has relied more on commercial-off-the-shelf (COTS) and contracted software systems. Government contractors and commercial developers currently rely heavily on semi-formal methods such as the Unified Modeling Language (UML) in developing the models and requirements for these software systems. The correctness of specifications in such languages cannot be tested, in general, until they are implemented. Due to the inherent safety requirements for mission critical systems, formal specification methods would be preferable. This thesis contrasts the development of a combat system for the Navy using the formal specification language SPEC with development using the semi-formal method UML. The application being developed is a ship recognition system that utilizes image data, detected emitters, and ship positioning to correlate ship identification. The requirements analysis and architectural design for this system are presented.

THIS PAGE IS INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION

A. NEEDS OF THE NAVY

The changing face of naval warfare in the modern age consists of littoral environments with multiple fast-moving ships and patrol boats. The ability of a ship commander to quickly and accurately identify contacts of interest (COI) greatly enhances his ship's survivability and mission effectiveness. With the advent of the Light Airborne Multipurpose System Mark III (LAMPS MKIII), the capability of U.S. Naval surface combatants was improved. LAMPS MKIII involves an SH-60B Seahawk helicopter and a LAMPS MKIII capable ship. LAMPS MKIII is configured to specifically aid ship commanders in controlling their battle space. The Seahawk extends the search and attack capabilities of a LAMPS MKIII configured destroyer, frigate, and cruiser platforms.

Traditionally the SH-60B LAMPS MKIII was used for undersea warfare (USW) missions and remote targeting for ship missile launches against enemy ships. However in 1997, with the addition of the AN/AAS-44V Forward Looking Infrared (FLIR) and the deployment of the SH-60B Rapid Deployment Kit (RDK) aircraft, the ability to accurately target and engage ships was added to the LAMPS MKIII mission. This added another primary mission, Surface Warfare (SUW), to LAMPS. With the advent of a FLIR and the AGM-114 Hellfire Missile, the SH-60B was now fully equipped to detect, track, and engage enemy targets while leaving the home ship undetected.

The normal method of identifying a COI is by passing a verbal description of the COI down the Hawklink to be analyzed by ship personnel. This method of identifying targets proves very inaccurate and, at times dangerous, due to the proximity of the helicopter to the COI. The AN/AAS-44V FLIR allows the helicopter to stand off from the COI and get a clear picture via the on-board video monitor. The FLIR video is now downlinked via the SH-60B's Hawklink to its home ship. This video can be analyzed by trained observers on the ship to identify COIs. While this has increased the information available to ship commanders, the ability to accurately identify COIs still relies on the capabilities of the human observer. If a computer were used to compare the given FLIR video to a database of known ships and provide identification, the commander's confidence in his knowledge of the battle space could be augmented.

Other data is also provided to the commander to help make his decision on whether a COI is hostile. The SH-60B has the ability, via the AN/ALQ-142 Electronic Support Measures (ESM), to detect emissions from COIs. The data collected via the ALQ-142 is down-linked via the HawkLink to the home ship. The home ship then uses the AN/SLQ-32 ESM system, the U.S. Navy's standard radar threat detection and analysis system, to analyze the information. An ESM operator (ESMO) controls the databases and operation of the SLQ-32 and works in coordination with the SH-60B's sensor operator (SENSO) to identify detected radars.

Finally, the U.S. Navy has deployed the Joint Maritime Command Information System Common Operating Environment (JMCIS COE) to its ships. This new system evolved from the Joint Operational Tactical System (JOTS); while some ships in the fleet still operate with JOTS, the upgrade to JMCIS is underway. These systems provide battlespace management. The JMCIS COE consists of communications, message processing, track management, tactical display, and validated applications. With JMCIS the commander is provided with a display of ship locations and identifications. It is just as important for the commander to know where his allied ships are located as the enemy vessels.

In modern naval battle, the intermixing of friendly and enemy ships is very probable. What is required to engage an enemy vessel in close proximity to friendly vessels is the ability to quickly and accurately identify COIs and then to surgically remove them from the battle field. The SH-60B RDK, and its successor the SH-60R, provides the latter. With the addition of a LASER designator and the AGM-114 Hellfire missile, the Seahawk helicopter has the ability to accurately target and destroy enemy ships.

The weakness of the entire approach is the identification of COIs. An automated method for FLIR image identification along with utilizing ESM and position data would greatly enhance this identification. The Maritime Analyzer Recognition Knowledge System (MARKS) is evolved from this concept.

MARKS is a non-cooperative target recognition system (NCTR), which does not rely on an interrogation/response process that is evident in an Identify Friend or Foe (IFF) system. [1]. An NCTR system takes inputs from one or more sensors and the COI is identified by means of data-fusion and target recognition algorithms. In the case of MARKS the sensors that will be used are the AN/AAS-44V FLIR, AN/ALQ-142 ESM, AN/SLQ-32 ESM, and JMCIS. While a database of emitters is available through the ship-based SLQ-32, an entire database of ship images, operating areas, and possible emitters is required to ensure the accuracy of MARKS. The use of passive methods of data collection allows for the home ship and LAMPS MKIII to remain undetected during the identification phase. While many current systems under use or being developed rely primarily on target emissions for identification, the addition of identifying a COI via its FLIR signature will increase the reliability of ship identification.

B. SOFTWARE ENGINEERING

Development of mission-critical software systems for the DOD has become a costly effort. With the reduction in defense spending, the need for cheaper and faster development of software systems has risen. The military's reliance on contractors to perform the analysis and design of many of this software has produced systems that do not meet the needs of the fleet. A review of methods for developing software shows that there are many different ways to tackle this problem.

While business systems and commercial software allow some degree of faults, as exemplified by many commercial products, military combat systems do not allow for faults of this degree. Fault tolerances in military software systems are required to be higher than the usual due to the inherent danger of having these systems fail. Plainly put, when military combat systems fail, people can die. The need for explicit and unambiguous specification of these systems is evident. While NASA, an organization that relies on highly fault-tolerant systems, continues to examine the use of formal methods, the DOD has fallen behind in this pursuit [2][3][4].

Commercial software companies can rely on a business model that

1) Release software that has been tested but is not complete;
2) Allow the millions of customers "test" their software and report back the bugs; and
3) Release software patches and updates via the Internet to fix the bugs.

The DOD does not have this luxury. The software developed for the DOD is not installed on nearly as many systems so the per capita cost is considerably more and errors in these software systems can prove to be even more costly. The Navy requires that combat systems be error free to a high magnitude. To achieve this DOD software systems are exhaustively tested. However, thorough testing can only reveal the presence of errors, it cannot guarantee the software's correctness.

The DOD relies on natural language descriptions of the requirements for the systems that are contracted. This type of description is inherently inexact due to the interpretability of the English language.

A methodology of describing the requirements for a software system must be applied to ensure clarity. The methods used to develop software for the DOD vary with contractors. So we explored two more precise methods in this thesis. An introduction to the two methods under review follows.

1. SPEC – A Formal Method

There are several interpretations of what a "formal method" is, but, generally, it is a method based on sound mathematical basis, typically seen in the method's formal specification language [5]. The formal specification language will have a well-defined syntax and semantics and a method for analyzing the specification produced [6]. The use of formal methods allows the software engineer to identify ambiguity and inconsistency in system design. The strength of a formal method lies in the ability to unambiguously define what a system does.

SPEC is a formal language for writing black-box specifications for software systems [7]. Figure 1.1 shows a diagram of the SPEC method. SPEC was developed to allow software developers to use a formal method for specifying large, complex systems. The base of the SPEC language is predicate logic. This provides for black and white specifications to be written [8].

SPEC was designed with the needs of large-system and real-time system developers in mind. Developers of these systems need to:

1) Localize information and isolate the details of an operation;
2) Represent generic concepts and make them available for re-use;
3) Detect and report interactions between system parts; and
4) Allow for the specification of timing constraints on the system.

SPEC addresses these issues through the use of concepts, inheritance, and the event-model [8].

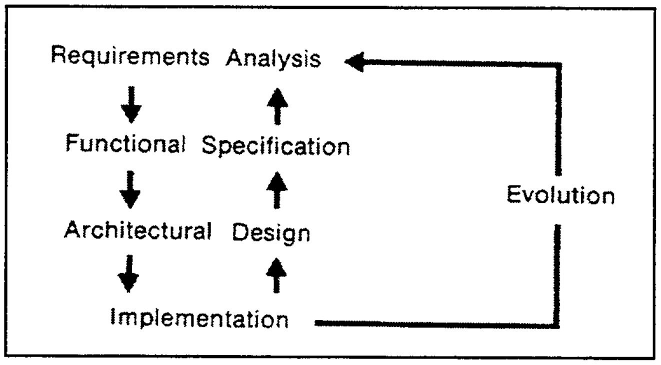


Figure 1.1 : SPEC Method [From 19]

The event-model of SPEC allows the developer to specify a module's behavior independently of its internal structure. SPEC deals with the *what* instead of the *how* of a module. This type of black-box specification along with the predicate logic base of the language allows for unambiguous specification of system operations and interactions.

While there are many other formal methods to choose from, Z (pronounced "Zed") being the most popular in Europe, SPEC was choosen because of the author's familiarity with developing software systems using the SPEC model. Many case studies performed on the use of formal methods have developers using a method for the first time. Generally much energy is spent on agonizing over the syntax of a language (or extending the syntax of the language to meet the needs of the project). An example of this is Martin Neil's attempt to specify a COBOL parser using Z where the syntactic rules and the general scope of Z were not followed, resulting in a poorly developed parser [9]. The problem with this is that the analysis process was tainted due to the developer's lack of knowledge. Although SPEC does not come with near as rich a set of tools as Z or many other formal methodologies, we did adhere carefully to the syntax, semantics, and method prescribe by SPEC.

Some might argue that formal methods are completely different and that by choosing one method the study would be biased to that type of method. However, in a practical sense, formal methods differ very little. A formal method allows a developer to express the ideas of a software system in a precise mathematical model [5]. In this sense a method like Z differs only slightly from SPEC.

2. UML – A Semiformal Method

Many of the popular methods for developing software systems are semiformal. To enhance the ability to perform stepwise refinement during the software development process, methods have been developed to provide abstract graphical representations of the system being developed. These methodologies are semiformal in that they attempt to accurately represent a software system by some systematic methods. However, the system model usually cannot be used to verify or predict a vast majority of system characteristics due to lack of formal semantics [10]. Although applying semiformal methods can aid in refinement of the system being developed, it cannot guarantee correctness of the system due to the lack of precise semantics of the specifications.

Over the past few years, UML has emerged as a popular semiformal method for developing software systems. UML provides a set of graphical and textual tools that allow the construction of an easily understandable model of a software system [11]. The graphical tools supplied in UML allow a software developer to easily model complex systems. While UML is generally related to object-oriented programming, it has facilities for many other aspects of software engineering process as well. (Figure 1.2)

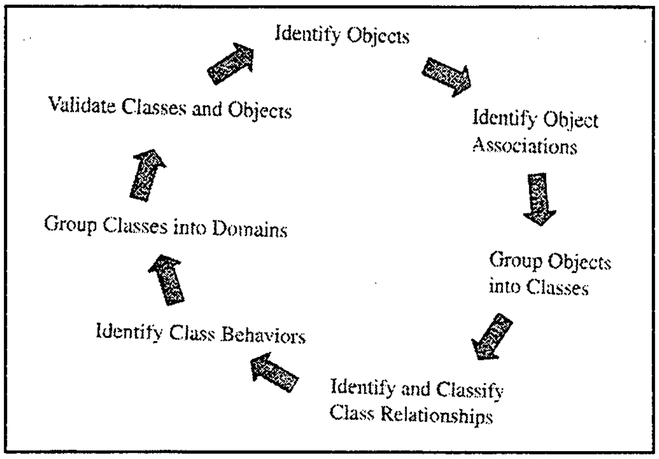

Identify Objects
Identify Object Associations
Group Objects into Classes
Identify and Classify Class Relationships
Identify Class Behaviors
Group Classes into Domains
Validate Classes and Objects

Figure 1.2: UML Method [From 20]

C. THESIS OVERVIEW

This thesis develops the requirements and model of MARKS using differing modeling methods. Comparison of the methods should not only fully flesh out the system but also provide insight into several critical areas of requirements analysis and system specification. We first provide a review of current technology for the NCTR, our test application. The focus of this thesis is two-fold.

- Develop MARKS as an NCTR for LAMPS MKIII ship commanders.
- Provide a comparison of developing mission critical systems using a formal method and a semiformal method of software engineering

The analysis of requirements and modeling of MARKS should provide the U.S. Navy with a well-designed plan for development of this system. Also, the review of software methodologies should provide insight into the need for requiring a standardized method for developing software for combat systems.

II. PROBLEM HISTORY

A. OVERVIEW OF NAVAL TARGET IDENTIFICATION METHODS

The Navy has generally relied on the method of visual identification (VID) of a target. Prior to engagement of a target, commanders will require a trained observer to identify a TOI using existing intelligence data and comparing the visually acquired TOI with that data. As stated above, in the realm of LAMPS MK III capable ships, this will usually require the SH-60B to fly within weapons range of the TOI. The aviation tactical officer (ATO), who is onboard the SH-60B, will provide the anti-submarine tactical air controller (ASTAC) a verbal description of the TOI [12]. The description provided is then checked against known ship descriptions provided by intelligence data. The method of VID of a target has produced mixed results. If the Pilot, ATO, and SENSO operating in the SH-60B are well trained and alert, and the ATO provides a good description, then the identification is generally accurate. However, the helicopter aircrew is required by this method to put themselves into harm's way to get the VID. Another result of VID of a TOI is that the target is then alerted to the presences of a LAMPS MK III capable ship. The element of surprise is lost and the probability of a kill is thus diminished.

Currently, the only NCTR available to the LAMPS MKIII ship/air team is the ship-based SLQ-32 working in conjunction with the helicopter based ALQ-142. The SLQ-32 provides a single, world-wide threat database in which radar modes are tagged with the specific region in which the radar mode operates. [13]. The ESMO can then select the appropriate Geo-area for analyzing emitter data. While the SLQ-32 contains an enormous library of emitter data and radar modes, there are many areas where it does not provide very accurate NCTR. The angle of error for detected emitters for the ALQ-14 is ±7 degrees. This alone could lead some to identify the wrong COI to the wrong emitter and thus misidentify a target. The detection of emissions from other ships for NCTR relies on the idea that ships maintain the same types of radar. Of course, this is not true. Ships rely on deceptive emissions and tightly control their emitters. Once again, reliance on emissions could result in misidentifying a TOI. Ship commanders who have come to understand this will require a VID and will not engage a target based solely on identification by emissions.

When operating in the current naval environment, it is especially important that commanders know where their fellow ships are located. As stated above, JMCIS provides this data. The identification of "friendlies" is very important in reducing the amount of blue-on-blue engagements. Blue-on-blue engagements are when friendly or allied combatants fire on one another. While JMCIS has come along in providing accurate battlefield data to ship commanders, there is still room for error in identification based on ship location. The littoral environment provides a battlefield full of small, mobile ships operating within the battle group. Therefore, if contacts "merge" due to their proximity there may be confusion with respect to which contact is actually friendly.

The commander needs the ability to merge all this data. If emitter data from the SLQ-32 could be checked against JMCIS ship data, and then augmented with identification via analysis of the ALQ-142 FLIR, ship commanders could have an accurate NCTR that enables the home ship to maintain the element of surprise while keeping is airborne SH-60B out of harm's way.

B. SOFTWARE DEVELOPMENT

During the early development of modern military systems, the primary challenge was to produce weapons and systems where the hardware involved was not prohibitive in cost. The software developed to run on these systems was a small portion of the overall product. In the end, the total cost of the project relied heavily on the cost of the hardware and not the software. However, beginning in the early 80s through today, advances in hardware development and microelectronics lessened the cost to develop hardware. The power of yesterday's multi-million dollar mainframe is available now on a single integrated chip. With the advances of software-aided hardware development, a dramatic decrease in hardware development cost has been evident. This has caused a shift in the costs to develop military systems. Today, a majority of the cost of military system lies with the development, implementation, and maintenance of the software.

1. Defining the Process

The current spiraling high costs of software developed for the military, along with extremely long development times and unpredictable quality of product, indicates that the DoD has problems in software

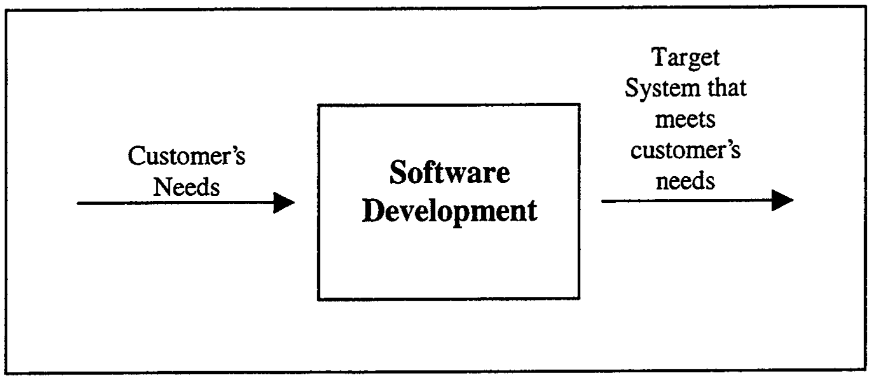


Figure 2.1: Software Development ≠ Computer Programming

development. In the early systems, programming was viewed as an art form instead of an engineering effort. Few formal methods existed and even fewer were used then [14]. Programming was basically a trial and error process via an undisciplined method. But more structured methods of developing software have evolved. The primary mistake of many military acquisition professionals was assuming that developing software is much like developing hardware. Where hardware relies on the laws of physics, no such laws apply to software development. As the science of computers moves towards an engineering discipline, these laws have to be discovered. The science of software engineering is at the stage of development that the science of physics was when Newton first wrote down his principles of physics. We are still discovering what constitutes software engineering and are laying down the laws that should be followed when developing software. What we do understand is that computer programming is not software development. Computer programming is developing a program that meets the needs of the programmer, while software development is producing software that analyzes the customer's needs and builds a target system that meets those needs, see Figure 2.1.

To that end, a definition of software engineering should be presented. Fritz Bauer at the first major conference on software engineering defined it as:

> *The establishment and use of sound engineering principles in order to obtain, economically, software that is reliable and works efficiently on real machines*[14].

It is interesting to note that Bauer introduced this definition in 1969 well before the advent of the cheaper hardware of today. Bauer highlights the need for managing the economy of software development. He understood, at the time, that the cost of developing software would rise as the systems became more complicated. He also highlights the need for reliability of software. When software has problems in the military it can lead to serious injury and even death. The IEEE furthers explains this in its definition of software engineering:

> *The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software* [14].

This definition suggests a development approach that can be reproduced. The waterfall model described the first systematic method of developing software. This method is shown in Figure 2.2. One of the myths of software development is that because software is flexible, change can be accommodated easily. While it is true that software requirements change, or evolve as the development cycle continues forward, the impact of the change varies with where in the development cycle one is. Its major problem is the difficulty in accommodating change after the process is underway.

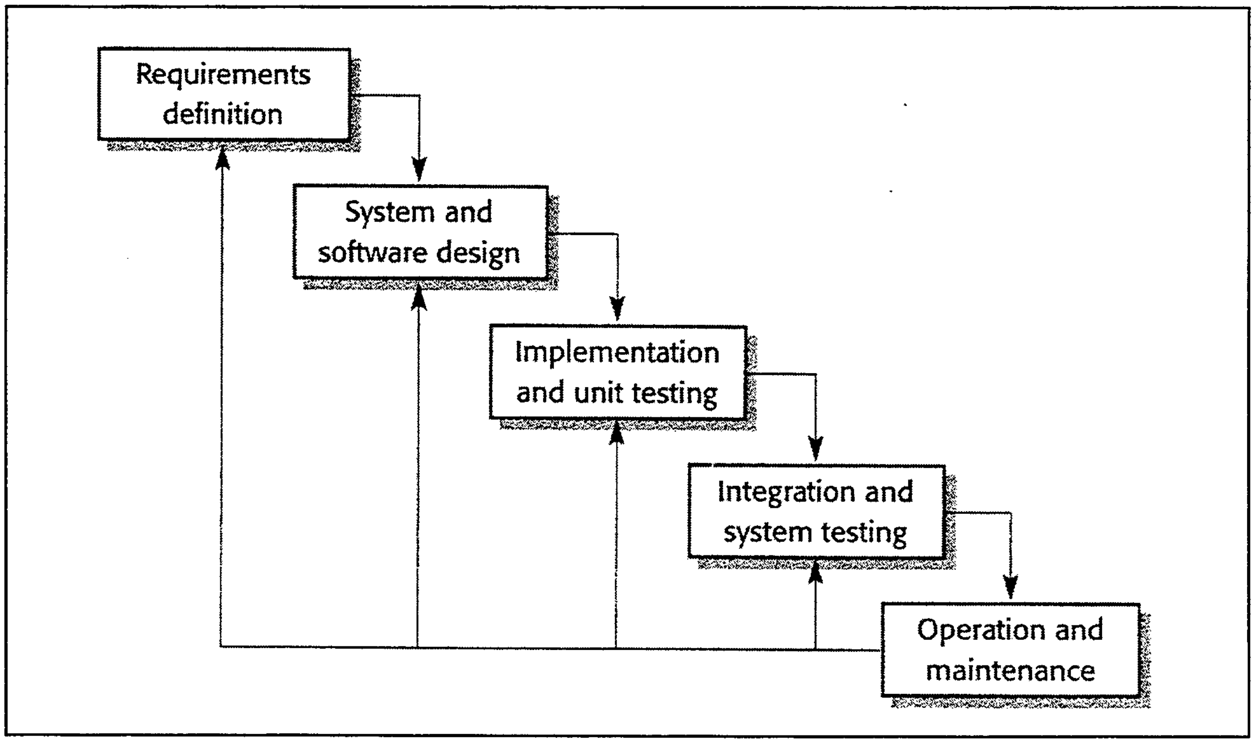

Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Figure 2.2: The Waterfall Model [From 20]

The current model for software development is the typical spiral model shown in Figure 2.3. This model incorporates the understanding that requirements may change during the process. It also provides risk analysis during the development cycle. With this model the move towards a more quantifiable approach to software development is evident. The model requires planning specific objectives for each

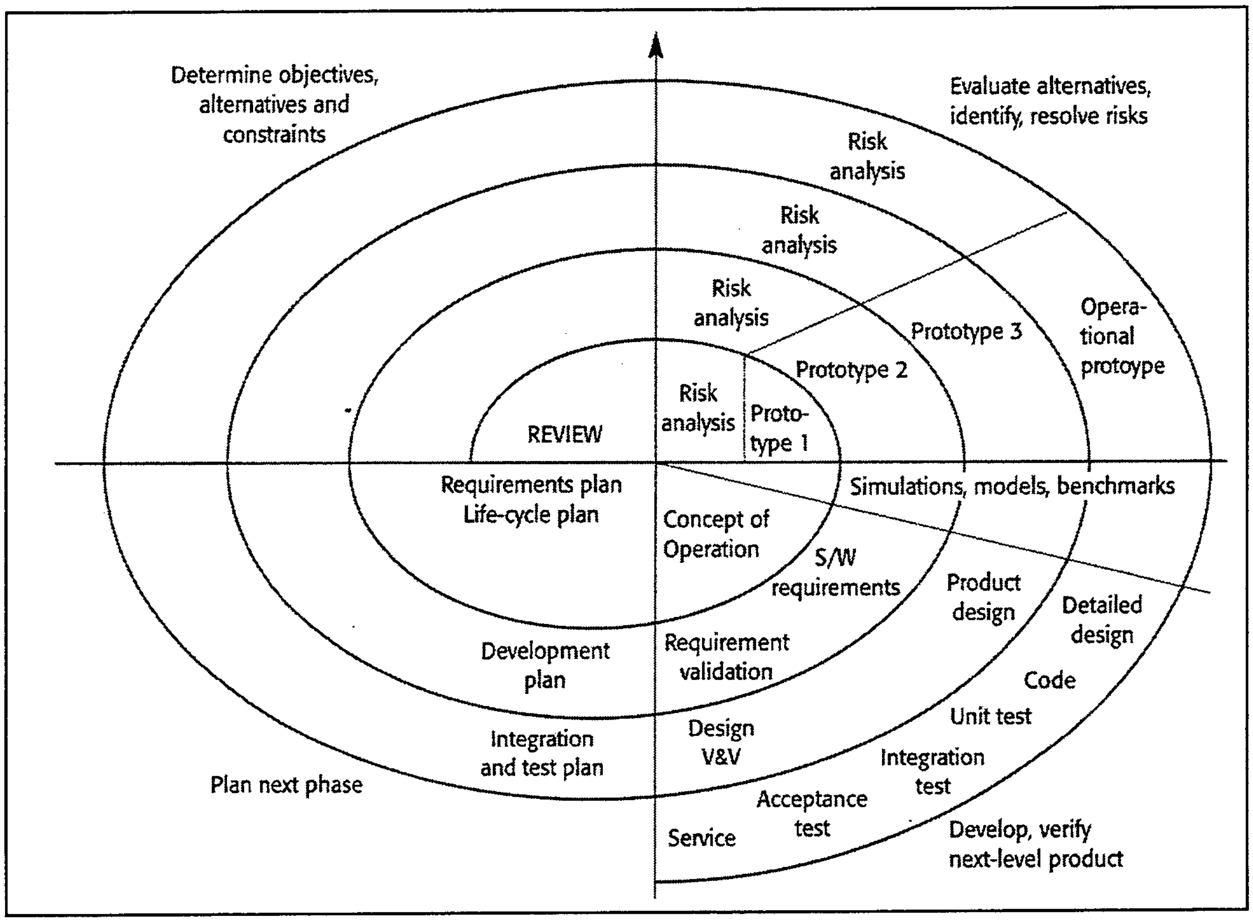


Figure 2.3: The Typical Spiral Model [From 20]

phase and identifying key risks to reduce these risks. An appropriate model is chosen for the next phase of development and the project is reviewed with plans being drawn up for the next spiral.

2. Further Development of the Methodology

The spiral method addresses the transition from a customer's statement of needs to traceable requirements that can be used to validate the system developed. In the development of military systems, the DOD serves as the customer. Generally, software is contracted out to civilian commercial software developers. It is the program manager's responsibility to ensure that the software developer under contract has a thorough understanding of the requirements for the software being built. Initial poor definition of software requirements is the major cause of failed software efforts [14]. The DOD must provide the contractor with a detailed description of the requirements for the system being developed. This can only be accomplished if a well-defined process is followed in development of these requirements and system models.

While increasing interest in a systematic approach to software development has been seen, more is needed as is shown when we look at the costs of failure. The amount of money spent on failed software systems is quite staggering. It has been estimated that $81 billion was spent on canceled software in 1995 and $100 billion was spent in 1996 [15]. In today's cost-cutting, money-conscious military, that amount of waste is unacceptable. Recently, the Undersecretary of Defense (Science & Technology) addressed this issue by stating in a memorandum

> *It is the DoD policy that software systems be designed and developed based upon software engineering principles...It also requires a software measurement process to plan and track the software program, and to assess and improve the development process and associated software product* [16].

This memorandum states the DoD's need for reliable and cost-effective software systems. In this thesis, the development of a combat system using differing methods of software development and a review of these methods should aid in pushing the DoD towards better software engineering methods.

3. Initial Comparison of the Software Modeling Methods

While SPEC has a deliberate process for developing software systems, UML does not. UML allows the developer to attack the problem from multiple directions while building a model of the proposed system. A comparison of the methods used in development in SPEC and UML are presented in Table 1. The popularity of UML may be attributed to this "loose" approach to software development. It gives the software engineer much freedom to manipulate the problem as he/she sees fit. However, inconsistency can arise in a UML model. These inconsistencies can be difficult to uncover early in the development process.

SPEC, on the other hand, stresses the adherence to a formal method. Its goal is to produce a consistent and unambiguous model of the system being developed. The benefit of dealing with a formal method is that, generally, errors in specification of the system are found earlier rather than later in the process.

While UML is readily understandable to the casual observer, and customers appreciate the graphical nature of UML's presentation, the model developed can be interpreted in many different ways. This is a problem when developing combat systems. The need for clear-cut specifications suggests a more formal method of producing software. However, formal methods are notorious for their cryptic syntax and

SPEC does not shy away from this. As SPEC is grounded in predicate calculus, an understanding of predicate logic is required to understand the specifications generated by SPEC. It is certainly not customer-friendly.

The focus of this thesis will be on analysis and design of an NCTR for the Navy. As the system is developed, the model properties outlined in Table 2.1 will be presented. The strengths and weaknesses of SPEC and UML will be discussed with a direct comparison of the models. In the end it is shown that adherence to sound software engineering practices produces a clear set of requirements and an unambiguous model of the system.

Process	UML	SPEC
Analysis	• Use-cases • Conceptual Model • Context Diagrams • Sequence Diagrams / System Operations Contracts	• Environmental Model / Inheritance Diagrams • Goal Hierarchy • Constraints • Functional Specification
Design	• State Diagrams • Collaboration Diagrams	• Message Flow Diagrams • Machines / Functions / Types • Dependency Diagrams • Transition Diagram • Stimuli and Response Diagram

Table 2.1 Comparison of Two Software Modeling Methods

THIS PAGE IS INTENTIONALLY LEFT BLANK

III. PROBLEM STATEMENT

A. THE INITIAL PROBLEM STATEMENT

The first step in any software development process is to meet the customer and review his software needs. In the case of this NCTR, the customer is a surface warfare officer (SWO) that will use the system. Discussions with several SWOs elicited this initial problem statement:

> *The purpose of the Maritime Analyzer Recognition Knowledge System (MARKS) is to identify contacts of interest (COI) providing a confidence meter reading. This is accomplished by analysis of downlinked Forward Looking Infrared (FLIR), Electronic Support Measures (ESM), and Joint Maritime Ship Information System (JMCIS). The system will allow the user to select a specific FLIR image to analyze, specific COI, and coordinate for JMCIS.*

The initial problem statement produced is easily understandable to the customer. Further analysis would be required before a software engineer not versed in the acronyms and terminology of the surface navy could develop this software.

Several meetings between the software developers and customers proved fruitful in breaking down the initial problem statement into an initial set of goals for the system. Communication between developer and customer is critical at this juncture of the process. The developer also needs to come up with questions that will lead to a basic understanding of the problem presented. As presented above, the explanations of COI, FLIR, JMSIS, ESM, etc. came out of these meetings.

A rudimentary set of goals was developed during the initial meetings with the customer, presented in Figure 3.1. These high-level goals defined the scope of MARKS and moved us closer to an in-depth look at the requirements for the system.

G1. THE GOAL OF THE MARITIME ANALYZER RECOGNITION KNOWLEDGE SYSTEM (MARKS) IS TO IDENTIFY MARITIME CONTACTS.

- G1.1. MARKS should incorporate FLIR analysis into its formula for identifying maritime contacts.
- G1.2. MARKS should incorporate ESM analysis into its formula for identifying maritime contacts
- G1.3. MARKS should incorporate JMCIS into its formula for identifying maritime contacts.
- G1.4. MARKS should provide an interface to the operator.

Figure 3.1: High Level Goals of MARKS

B. FEASIBILITY ASSESSMENT

Before moving forward with either method of software development a question needed to be asked: "Is this project feasible?" It had to be determined whether a system could be developed with existing hardware and software engineering methods that could fulfill the customer's needs.

1. JMCIS

The current system used by the U.S. Navy, JMCIS, has been described above. Initially JMCIS was a UNIX-based system. The core of the original incarnation of JMCIS was the X-Windows system in the UNIX environment [17]. The original JMCIS was developed with extensibility issues in mind. Therefore, other UNIX-based hardware and software systems should not have a problem integrating with the JMCIS COE.

The most recent JMCIS COE is being developed for PC-based systems. Labeled JMCIS 98, this incarnation will incrementally bring UNIX-based functions to the PC environment. With the advent of IT-21, the U.S. Navy is moving rapidly to a common PC-based operating environment [13]. The combination of IT-21 PC-based local area network (LAN) and the PC-based JMCIS 98 COE will provide for easy integration of MARKS with JMCIS.

2. ESM Analysis

The SLQ-32 is the standard method for performing ESM analysis on U.S. Naval warships. While the system is proprietary and developed by Raytheon Systems, there are several variants on ships worldwide. The base computer for the SLQ-32 is the AN/UYK-19 central computer. Along with an emitter file memory subsystem, the SLQ-32 can passively identify targets based on electronic emissions. Several upgrades have been developed for the SLQ-32 system. The SLQ-32A(V) has been introduced for a new build of the system. This new system reflects the scope of architectural and processing developments introduced over the last decade [17]. The SLQ-32A(V) also comes with a PC-based trainer. This trainer synthesizes and injects simulated, programmable emitter contact signals into AN/SLQ-32A EW equipment in support of dockside or at-sea training exercises [13].

The development of add-on systems shows that, even though the base system is proprietary, it is possible to integrate with the SLQ-32. MARKS would not be required to communicate with the SLQ-32 database. The only data required would be identified emitters by the SLQ-32. This would require no signal processing on MARKS' part. In the end, while not as simple to integrate with as JMCIS, we determined that MARKS could communicate with the SLQ-32.

3. FLIR Analysis System

While integration with JMCIS and the SLQ-32 is the overriding problem for dealing with ship positions and emitter data, that is not the case with FLIR analysis. The downlinked FLIR video from the deployed LAMPS MKIII helicopter is displayed on a standard television set. The video is linked via a simple coaxial cable. This approach allows the user to connect the coaxial cable to a videocassette recorder (VCR) for recording and later playback. Therefore, getting the information from the LAMPS MKIII proved to be quite easy. Interpreting that data is harder, however.

First generation FLIRs provided very poor picture quality to the user. To enable identification even by human means the operator must be in close proximity to the target. While it did give the operator the ability to see more at night, it did not provide enough data for clear identification. The AN/AAS-44V FLIR, a second generation FLIR, provided the clarity and standoff ranges needed. This FLIR has the ability to display 1/100th of degree temperature differentials. The clarity of these images is greatly increased over that of the first generation FLIRs.

A study was initiated to determine whether we could actually identify given images. The initial study required that the FLIR image of the ship to be identified be broadside with the ship filling a majority of the screen. LT Jessica L. Herman, USN, developed an algorithm for identifying FLIR images of ships [18]. Her research proved that with the given limitations of current hardware and software we could fully develop a FLIR image analysis system. Subsequent work by LT Jorge Alves is extending this work to arbitrary angles of view.

4. Overall Feasibility

The feasibility study showed that further development of MARKS should continue. This thesis will now show the development of MARKS under SPEC in Chapter 4 and UML in Chapter 5. The development under this two models should highlight many other goals and requirements for MARKS. As the system is developed the continued evaluation of the feasibility should be considered as the goals and requirements are outlined.

IV. MARKS IN SPEC

A. REQUIREMENTS ANALYSIS

The purpose of requirements analysis under SPEC is to take the customer's initial problem statement and convert it to a precise statement of the requirements (see Figure 4.1). The initial problem statement provided by the customer is in natural language and is generally informal, abstract, ambiguous, incomplete, contradictory and contains no cost/schedule considerations [19]. Looking at the problem statement for MARKS given in Chapter 3 we thus see that it is an incomplete statement of requirements for MARKS. Further development of the goals and requirements for MARKS was required prior to designing the system. The requirements document developed by using SPEC will turn the customer's informal problem statement into a precise, testable, and feasible set of requirements.

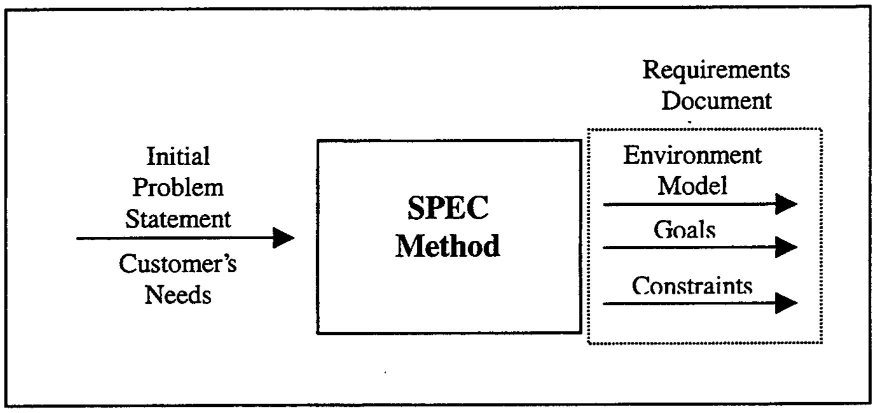


Figure 4.1: Requirements Analysis under the SPEC Method

1. The Environment Model

An environment model allows for communication between the customer and developer on system design and requirements. The environment model provides a formal vocabulary for describing the customer's problem statement. The concepts for which MARKS will operate must be defined first. A review of the customer's problem statement highlights the concepts that will need to be defined. The potential concepts include:

- MARKS;
- FLIR Analyzer;
- ESM Analyzer;
- JMCIS Analyzer;

- Graphical User Interface (GUI);
- FLIR;
- ESM;
- JMCIS;
- Confidence meter;
- COI;
- FLIR Image; and
- Coordinate.

However, we wanted to ensure that MARKS was not limited just to the ship and aircraft systems described

```
DEFINITION marks_definitions

  INHERIT station_definitions
  INHERIT sensor_definitions
  INHERIT image_system_definitions
  INHERIT emitter_analysis_system_definitions
  INHERIT ship_position_analysis_definitions
  INHERIT operator_definitions

  EXPORT marks

  CONCEPT  marks: software_system
    WHERE proposed(marks)
    --the goal of the system is to build MARKS

    & uses(operator, marks)
     --a human operator uses MARKS
    & uses(marks, image_analysis_system)
    & uses(marks, emitter_analysis_system)
    & uses(marks, ship_position_analysis)
END
```

Figure 4.2: Marks Definitions in the Environment Model

```
DEFINITION ship_definitions

 IMPORT Subtype from type

  CONCEPT ship : type

  CONCEPT ship_type : type
    WHERE Subtype(ship_type, ship)

  CONCEPT ship_class : type
    WHERE Subtype(ship_class, ship)


  CONCEPT valid_ship(s : ship) VALUE (b : boolean)
   --b if given ship is a valid ship
END
```

Figure 4.3: Ship Definitions in the Environment Model

above. As the military continues to upgrade and evolve its hardware, MARKS should change to incorporate those upgrades. We decided to extend a few of the potential concepts to capture a broader range of systems. In particular, FLIR, FLIR Analyzer, and FLIR Image were broadened to image system, image-analysis system, and image to cover video cameras, digital cameras, and other optical systems. Also, JMCIS and JMCIS analyzer were broadened to ship-positioning system and ship-positioning analysis system, ESM and ESM analyzer were broadened to emitter-system and emitter-analyzer system, and COI was broadened to ship. The ship-positioning system JOTS is still operational on older fleet ships so we are allowing MARKS to be backwards compatible to JOTS.

With the stated concepts, we modeled the environment for MARKS. This model represents a simplified view of the setting that MARKS will operate in; we need to model only the aspects of the environment that were relevant to the customer's problem statement [19]. This model included both aspects for MARKS and those external to MARKS. Figure 4.2 shows the definition of MARKS as a software system. It says that the system is a proposed system, who will use the system, and what other systems that MARKS will use. These concepts need to be defined to develop goals for the system. Then we defined even more simple concepts such as "ship" which was identified above. The SPEC for this is outlined in Figure 4.3. We broke out further concepts as we developed the environment model. Here we defined the concept of a ship but we also defined a ship type and a ship class. In identifying ships, humans generally will broadly classify it, e.g. merchant, military, fishing, etc., and then subclassify it, e.g. frigate, cargo, oiler, etc. With these definitions we incorporated the terminology of the customer so that it would be easier for them to understand the model.

Figure 4.2 shows other concepts in the environment model including sensor definintions, operator definitions, image system definitions, etc. These definitions can be found in Appendix A with the full MARKS Environment Model. The model seen is the final product after considerable analysis of the system. The developer, customer, and domain experts review the environment model. Before further development of MARKS could continue, all stakeholders had to agree on this environment model as it provides the basis for MARKS.

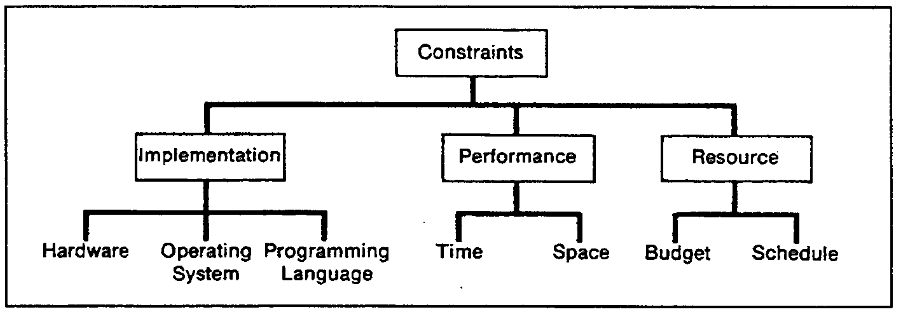


Figure 4.4: Constraints in Requirements Analysis [From 19]

2. MARKS Constraints

In conjunction with developing the environment model, we investigated the constraints of MARKS. In determining the constraints for a system, we needed to look at three types that are relevant to requirements analysis. These constraints are implementation constraints, performance constraints, and resource constraints, and they are shown in Figure 4.4. To implement the system it was determined that a PC-based system running Windows 9x/ME or Windows NT/2000 with 256 megabytes (MB) of random access memory (RAM), a 10 gigabyte (GB) 7200 millisecond (ms) hard drive, and a 21 inch 1200x1600 resolution monitor would fulfill the implementation constraints. The programming language was not specified during this process. The only performance constraint that could be identified was that MARKS would have to provide identification within 30 seconds. The development of MARKS in SPEC was determined to take 3 months with the feasibility study for image analysis taking 2 months concurrently.

3. MARKS High Level Requirements

While the high level goals shown in Figure 3.1 were enough to begin initial analysis of MARKS, a further development of the goals and requirements of the system was necessary. For MARKS to work correctly, the station using it must incorporate an imaging system, emitter system, and ship-positioning system. While MARKS is being developed to work on U.S. Navy ships, we did not want to restrict it usage to that. We identified the concept of a station, used it in the requirements, and defined it in the environment model (Figures 4.5 and 4.6). We defined "station" as a type where all stations must have as a part of them a ship position system, emitter system, and imaging system. We augmented the requirements for MARKS by stating that image analysis, emitter analysis, and ship position analysis should be incorporated in

G1. The goal of the Maritime Analyzer Recognition Knowledge System (MARKS) is to identify maritime contacts.

- G1.1. MARKS should incorporate an image analysis system into its formula for identifying maritime contacts.
  - R1.1.1. An imaging system should be a part of the station using MARKS.
  - R1.1.2. Image analysis must correctly identify a maritime contact giving a confidence reading.
- G1.2. MARKS should incorporate an emitter analysis system into its formula for identifying maritime contacts
  - R1.2.1. An emitter system should be a part of the station using MARKS.
  - R1.2.2. Emitter analysis must correctly identify a maritime contact giving a confidence reading.
- G1.3. MARKS should incorporate a ship-positioning analysis system into its formula for identifying maritime contacts.
  - R1.3.1. A ship position system should be a part of the station using MARKS.
  - R1.3.2. Ship position analysis should provide a list of ships located near the given coordinate.
- G1.4. MARKS should provide an interface to the operator.
  - R1.4.1. A graphical user interface should be utilized to convey information to the operator.
  - R1.4.2. A graphical user interface should be utilized to get information from the operator.

Figure 4.5: High Level Requirements Tree for MARKS

MARKS. The specification of this in SPEC leaves the customer with an unambiguous definition of what constitutes a station. Even as we analyzed the requirements we further explored the environment that MARKS would be operating in. Finally, it is specifically outlined that a GUI will be used to allow the operator to input and receive information from MARKS.

These high level requirements are very close to the customer's initial problem statement; they are still somewhat qualitative and are difficult to test. The need to refine the requirements is evident. The

```
DEFINITION station_definitions

  INHERIT system
  INHERIT sensor_definitions
  INHERIT relationship2

  CONCEPT station: type
    WHERE ALL(st : station :: part_of(ship_position_sytem, station)
     & part_of(emmiter_system, station)
     & part_of(image_system, station))
    --station will have ship_position_system, emitter_system, image_system

END
```

Figure 4.6: MARKS Environment Model Station Definitions

ultimate goal of refining the requirement is to develop a set of requirements that are testable by mathematical means. In general, as requirements are further refined, they get more formal [19].

B. FUNCTIONAL SPEC

During the functional specification, we take the output from the requirements analysis stage and define a black-box specification. We wanted to capture the behavior with respect to the users of the system. The goal of this stage of the process is not only to develop a complete set of requirements as shown in Figure 4.7, but to develop a description of the interfaces that MARKS will have with other systems and the user. We have captured the problem domain via the environment model and have laid out the goals and constraints of the system during the requirements analysis stage. Now we have to decide how MARKS will communicate with the outside world. At this point we did not delve into the inner workings of MARKS. We stayed at the level that MARKS is a black-box and had to model that black-box to meet the requirements set forth. During this phase we were still asking what it is we wanted to build and not how we wanted to build it. It was therefore prudent at this stage to define the system interfaces and actions of MARKS.

The functional specification we developed was explicitly set to define the behavior of MARKS with respect to abstract inputs and outputs. It is an abstract functional specification for MARKS. Concrete specifications that define specific formats and editing facilities for these inputs and outputs were not defined.

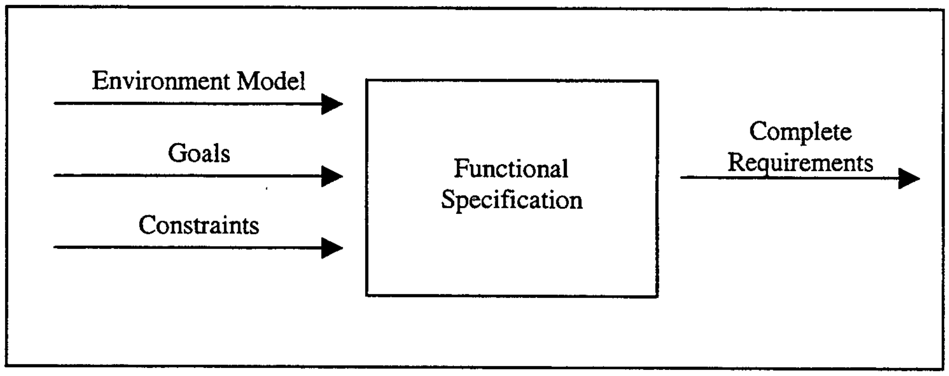


Figure 4.7: Input/Output of Functional SPEC

1. MARKS System Interfaces

The first step in developing the functional SPEC was to identify the external systems that would interact with MARKS. We examined the environment model along with the goal/requirements tree and concluded that MARKS would have to interface with

a. the operator;
b. an image system;
c. an emitter system; and
d. and a ship position system.

Figure 4.8 shows a graphical representation of the MARKS system interfaces that were modeled in the functional specification. Using Figure 4.8, we created a skeleton spec module for each of the external systems and the interfaces to those systems. Using the graphical representation and spec modules we examined the messages that would be required for MARKS operation.

To develop the interface with the imaging system, we first had to identify the messages that would be needed. The imaging system was modeled as a function as shown in figure 4.9. It simply states that when the imaging system gets a new image it sends that image via the "new_image" message to the image_system_interface. This "new_image" is a full motion image. Therefore, a type module for full_motion_image was developed, shown in Figure 4.11. The Type module in SPEC defines an abstract

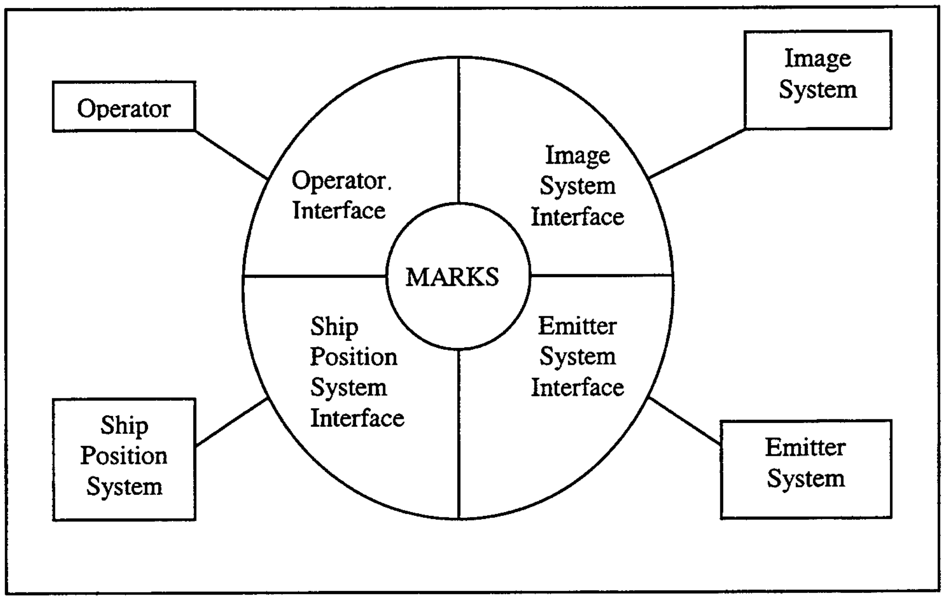


Figure 4.8: MARKS System Interfaces

```
FUNCTION image_system

  INHERIT image_capture_definitions

  MESSAGE get_new_image(fmi : full_motion_image)
    SEND new_image(fmi) to image_system_interface

END
```

Figure 4.9: Function for image system

```
TYPE full_motion_image

  INHERIT image_capture_definitions

   MODEL(full_motion_image : sequence{frame})   --A video full motion images consists of a set of frames
   INVARIANT   ALL(fmi : full_motion_image :: frame_rate(fmi) >= 25)

   MESSAGE update_image(fmi : full_motion_image)
      REPLY (fmi2 : full_motion_image)  WHERE fmi2 = images(fmi)
   --Given a full motion image display that full motion image

  MESSAGE freeze_image(fmi : full_motion_image, f : frame)
    REPLY(fmi2 : full_motion_image)  WHERE fmi2 = {f}
    --Given the current full motion image and a specific frame, just display that frame

  --Supporting Definitions
  CONCEPT images(fmi : full_motion_image) VALUE (f : set{frame})
  --The set of frames is defined as a full_motion_image

  CONCEPT image(fmi : full_motion_image) VALUE (f : frame)
  --given a full motion image this will return the current frame.
END
```

Figure 4.10: Type for the full motion image

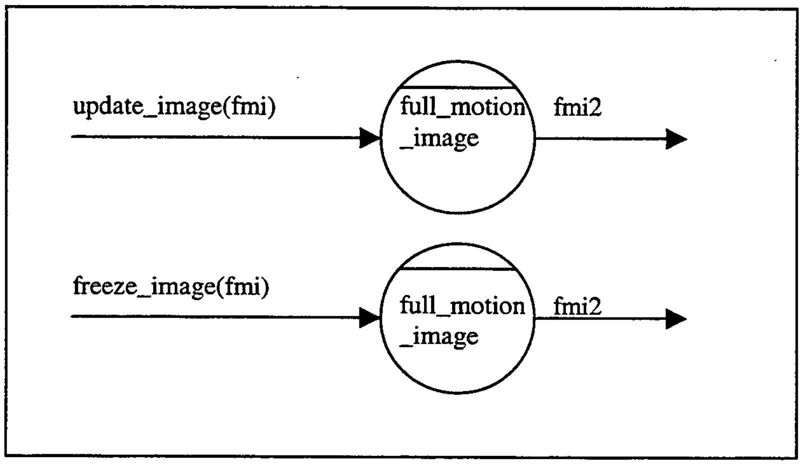


Figure 4.11: The stimulus-response diagrams for full_motion_image

data type [19]. The value set in this instance models a full_motion_image as a sequence of frames where the frame rate is greater than or equal to 25 frames per second. The operations that can be performed on a full_motion_image type are update_image and freeze_image with both of these messages taking a full_motion_image as an input. The stimulus-response diagram for full_motion_image, shown in Figure 4.11, gives a graphical description of full_motion_image message actions.

The interface to the imaging system is therefore fairly simple. The definitions for the base types of the image system and image system interface appear in the environment model developed earlier. We modeled the two messages that MARKS will require from the interface to the image system. These are update_image and freeze_image. Both are shown in the machine definition for the image system interface in Figure 4.12. In SPEC, a machine is a module that has an internal state [19]. The message new_image provides the only changes to the state of the image system interface. When a new_image message arrives the state of the image system interface is updated with this new full motion image, as shown in Figure 4.13. The other messages in this machine, update_image and freeze_image, serve to allow MARKS the ability to

```
MACHINE image_system_interface

   STATE (video : full_motion_image)
   INVARIANT fmi_on
   INITIALLY  SOME(fmi : full_motion_image :: new_image(fmi))

   MESSAGE update_image REPLY (v : full_motion_image) WHERE v = video
     --Full motion video returned is current full motion image held by image system interface

  MESSAGE freeze_image REPLY(f : full_motion_image) where f = freeze_image(video, f)
     --f returned is a full motion image of the same frame

   MESSAGE new_image(fmi : full_motion_image) TRANSITION video = update_image(fmi)
      --new full motion image sent to interface is accepted as current full motion image held by interface

   CONCEPT fmi_on : boolean
    WHERE fmi_on <=> ALL(fmi : full_motion_image :: frame_rate(fmi) >= 25 & image_system_detected)
     --full motion image is present if the frame rate is greater than 25 and the image system is detected.

   CONCEPT image_system_detected : boolean
    --if image system is detected then this is true
END
```

Figure 4.12: Machine for the image system interface

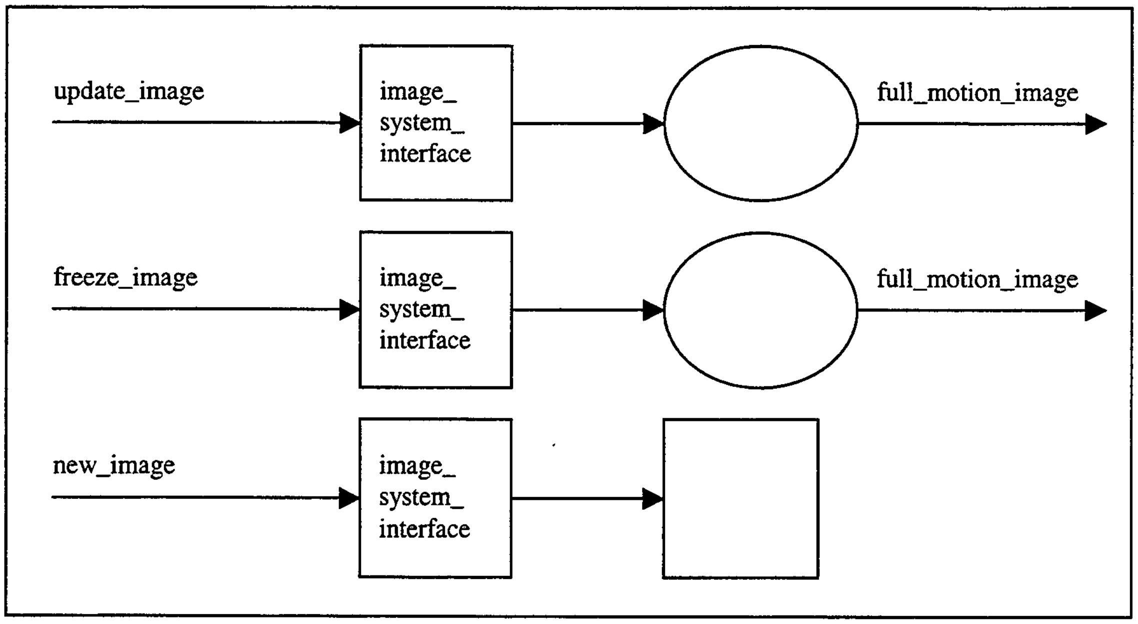


Figure 4.13: Stimulus-response diagrams for the image_system_interface

The other messages in this machine, update_image and freeze_image, serve to allow MARKS the ability to retrieve information from the interface.

The process of defining the external systems and interfaces between those systems and MARKS continued similar to what has been described above. The results of this are shown in Appendix C. In specifying the external systems and messages we also developed a simple diagram containing that shows graphically the message traffic to and from MARKS. The message flow diagram is shown in Appendix D.

To ensure that these requirements can be traced a formalized requirements module was also defined. As the model of MARKS evolved and the depth of the goal/requirements tree grew, the module containing the formalized requirements continued to evolve. The end result for the MARKS system goals is shown in Figure 4.14. In defining the concepts of this module, we ensured that all the leaf goals defined in the goal/requirements tree where traceable. The final output of the functional SPEC is the fully fleshed-out requirements tree. While the final goal/requirements tree is shown in Appendix A, it should be noted that it took several iterations of the functional specification to reach this.

```
DEFINITION marks_system_goals

  INHERIT requirement_goals
  INHERIT image_capture_definitions

  CONCEPT find_ship_type(image : frame) VALUE (sts : set{tuple
    {st :: ship_type, cm :: confidence_meter}})
    WHERE ALL(st : ship_type :: st IN sts <=> st.confidence_meter > 0.4),
              goal(find_ship_type,marks)  ---G1.1.1.2.2.1

  CONCEPT find_ship_class(image : frame) VALUE (scs :
     set{tuple{sc :: ship_class, cm :: confidence_meter}})
     WHERE ALL(sc : ship_class :: st IN scs <=> sc.confidence_meter > 0.4),
               goal(find_ship_class,marks)  ---G1.1.1.2.2.2

  CONCEPT find_ships_by_emitters(em : sequence{emitter})
                        VALUE (ships :  set{tuple{s :: ship, cm :: confidence_meter}})
     WHERE ALL(s : ship :: s IN ships <=> s.confidence_meter > 0.4),
               goal(find_ships_by_emitters,marks)  ---G1.1.2.2

  CONCEPT find_ships_by_position(c : coordinate, r : real)
                        VALUE (ships : set{tuple{s :: ship, cm :: confidence_meter}})
     WHERE ALL(s : ship :: s IN ships <=> s.confidence_meter > 0.4),
              goal(find_ships_by_position,marks)  ---G1.1.3.2

  CONCEPT marks_display : boolean WHERE goal(marks_display, marks) --G1.2.1

  CONCEPT have_image_capture_capability : Boolean
      WHERE image_capture_capability, goal(have_image_capture_capability, marks) --G1.2.2.1

  CONCEPT emitter_search_intialize : boolean WHERE goal(emitter_search_intialize, marks)
     --G1.2.2.2

  CONCEPT ship_search_initialize : boolean WHERE goal(ship_search_intialize, marks)  --G1.2.2.3

  CONCEPT display_marks_setup : boolean WHERE goal(display_marks_setup, marks)
END
```

Figure 4.14: The MARKS formalized requirements module

THIS PAGE IS INTENTIONALLY LEFT BLANK

V. MARKS IN UML

A. REQUIREMENTS ANALYSIS: A USE-CASE METHOD

For an object-oriented approach to modeling a system, UML has become the *de facto* standard [20]. We began our analysis of MARKS by doing a use-case analysis of the system. Figure 5.1 illustrates the desired results of the use-case analysis. We identified the systems and operators external to MARKS as the operator, the image system, the emitter system, and the ship positioning system. Therefore, we used UML to model the behavior of these external systems but we did not further develop these systems. By identifying the external factors to MARKS we also delineated the system boundary. We began by modeling the system using high-level use-case diagrams. These provide a context diagram for understanding how the operator will interact with MARKS [21]. In Figure 5.2, the actor on MARKS is

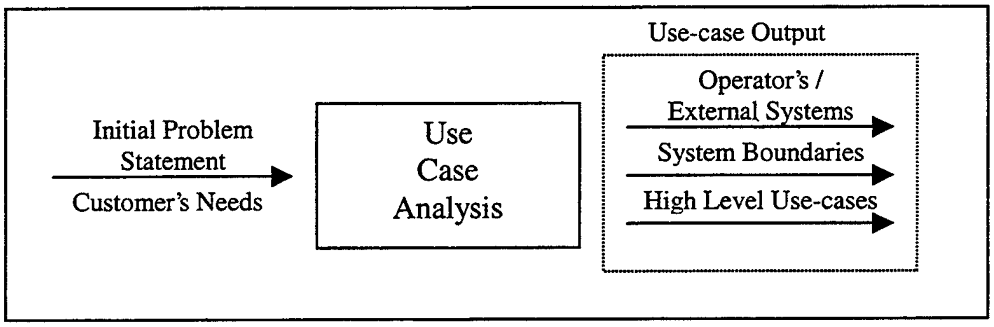


Figure 5.1: UML Use-case Analysis

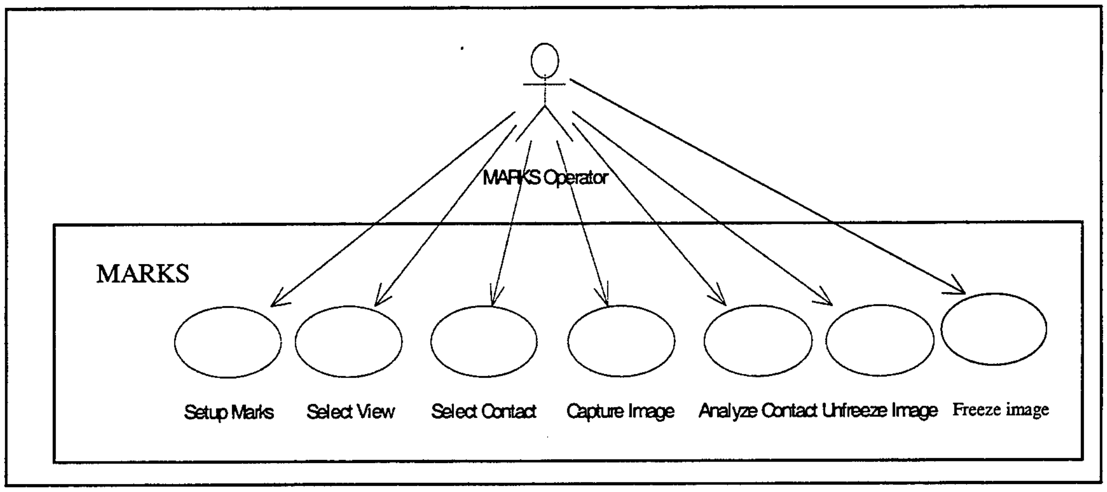


Figure 5.2: A high-level use-case for operator interaction with MARKS

shown as a stick figure with each class of interaction as ellipses. The box around the ellipses shows the system boundary of MARKS. It is important to delineate system boundaries when developing use-cases. External events that directly stimulate MARKS, such as the operator shown in Figure 5.1, are developed during this phase. We created several of these high-level diagrams, as seen in Appendix H.

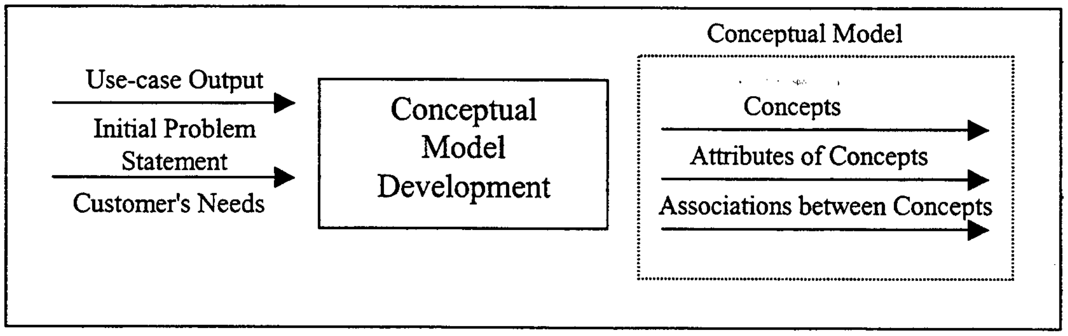

Use-case Output
Initial Problem Statement
Customer's Needs
Conceptual Model Development
Conceptual Model
Concepts
Attributes of Concepts
Associations between Concepts

Figure 5.3: Conceptual Model Development

B. MARKS CONCEPTUAL MODEL

After the use-case diagrams for MARKS, we developed the conceptual model. The conceptual model takes an object-oriented analysis of a software system and decomposes the concepts of the problem domain. When developing the conceptual model, we focused solely on domain concepts and not software entities. The output of this phase will be a model of the system consisting of concepts, attributes of concepts, and associations between the concepts as shown in Figure 5.3. We defined the structure of the concepts and not the operations that those concepts could perform.

Reviewing the problem statement given by the customer and the use-cases developed we identified the concepts required to develop the conceptual model:

- MARKS;
- Operator interface;
- Image system;
- Emitter system; and
- Ship-position system.

We further amended the conceptual model to show the differences in the operator interface and broke out what MARKS would require to identify a given COI. The additional concepts are:

- Analyze view;
- Ship-position view;
- Emitter view;

- Image view;
- Image system interface;
- Emitter system interface;
- Ship-position system interface;
- Image identification system;
- Emitter identification system; and
- Ship-position tracker.

The whole domain must be captured in the conceptual model. Therefore, as illustrated in Figure 5.4, systems external and internal to MARKS are modeled. Here we show MARKS' concepts and associations between its concepts. Lines connecting concepts delineate these associations. The diamond and line as seen in Figure 5.3 illustrates a composition. For example, in the conceptual model, the Image_ID_Sytem is composed of an Image_Type_ID and an Image_Class_ID. We decomposed MARKS into simplified concepts that allowed us to clarify the system. While Figure 5.4 shows the conceptual model, it is high-

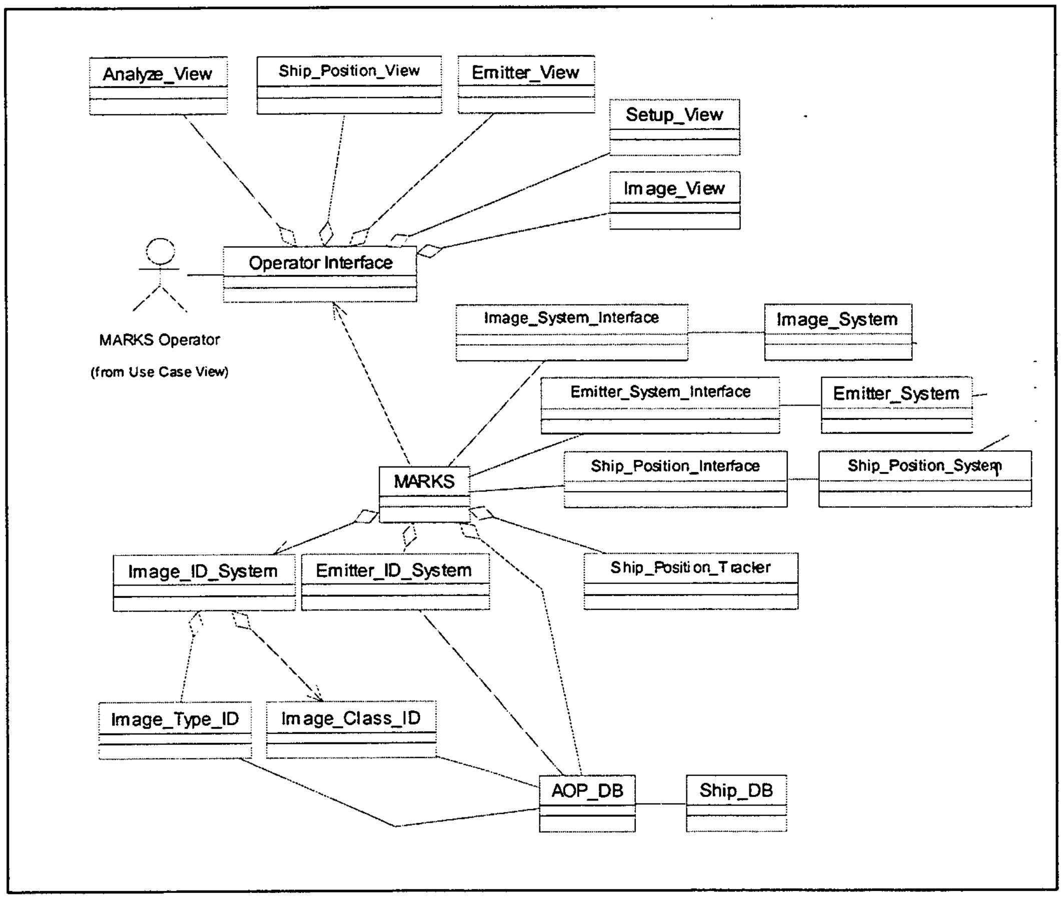


Figure 5.4: MARKS Conceptual Model

level. Several other models were built to show the relations of subsystems and to further specify the attributes of each concept.

C. EXPANDED USE-CASE ANALYSIS

The purpose of taking a second look at the use-cases for MARKS allowed us to provide more detail than the high-level use-case analysis allowed. This is required in order to obtain a deeper understanding of the processes and requirements for MARKS. Figure 5.5 shows the process of developing the expanded use-cases. The output from the previous use-case analysis, the conceptual model, and the initial customer problem statement are used to further develop the use-cases for MARKS. We developed

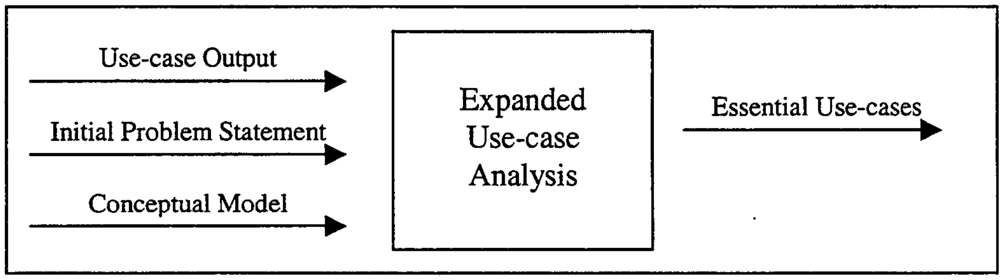


Figure 5.5: Expanded use-case analysis

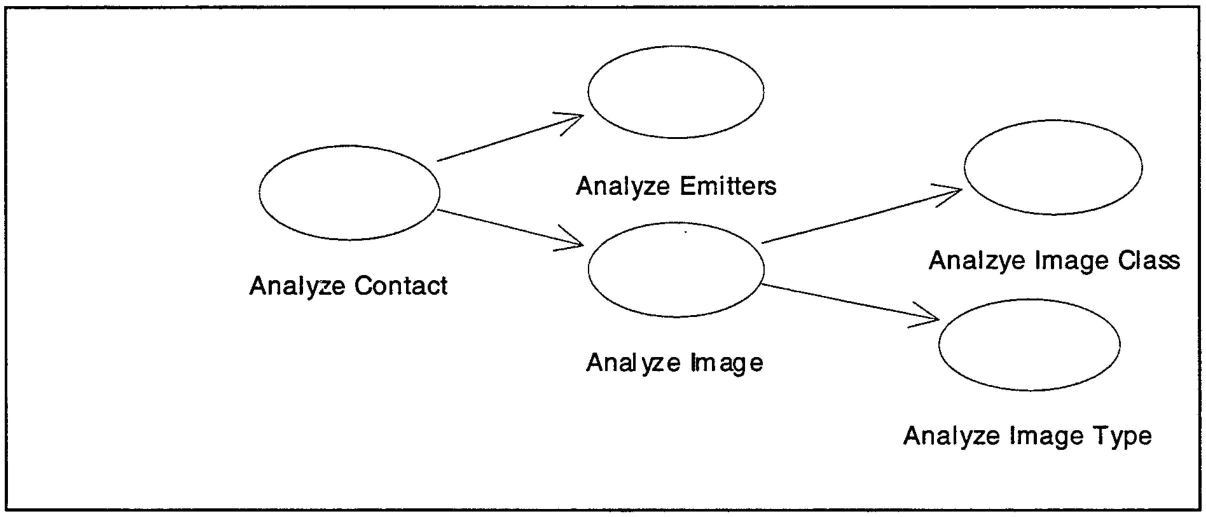


Figure 5.6: Analyze Contact low level use-case

lower-level use-cases as illustrated in Figure 5.6 the low-level use-case of Analyze Contact. The use-case Analyze Contact uses both the Analyze Emitters and Analyze Image use-cases. The arrows in the diagram show this.

An example of the expanded format for a use-case is shown in Figure 5.7. The expanded use-case format is a natural language description of the use-case presented. This use-case description shows the actors, preconditions, postconditions, and a simple description of the use-case. It is important at this point to capture the most important use-cases in this expanded format. The use-case diagrams and expanded use-

Use-case:	Analyze Image Type
Actor:	Image_ID_System
Precondition:	AOP is specified. A valid image has been captured to memory.
Description:	Image_Type_Id is passed the image of the ship to be identified in memory. The image is then converted to an image_type and matched against the ship image_type's from the respective aop and country databases. A list of matches with ship type and confidence meter is then returned.
Post conditions:	The Image_ID_System holds a set of matching image types with respective confidence meters.

Figure 5.7: Use-case for Analyze Image Type

Essential

Typical Course of Action

Image_ID_System Action	**Image_Type_ID Response**
1. Passes an image for analysis.	2. Converts the image to an image type.
	3. Obtains a list of ship types.
	4. Completes a type match of image against the list.
	5. Returns a list of ship types with associated confidence meters.

Figure 5.8 Essential use-case for Analyze Image Type

case descriptions were then shown to the customer for approval. Finally, the essential use-case was developed to describe MARKS' processes in terms of essential activities. It is important to describe the processes using implementation free details. An example of the essential use-case description for Analyze Image Type is shown in Figure 5.8. The essential use-case is in contrast to a concrete description of the system process. In a concrete, or real, use-case the process is described in terms of a current design, where a commitment to specific input and output technologies exists. We did not develop any real use-cases. The spectrum of use-cases can be seen as a degree of design commitment. This continuum is shown in Figure 5.9. While the high-level use-cases are considered essential use-cases due to their brevity, the essential use-case presented provides a deeper understanding of the process and is moving the design towards a the right on the continuum. As the use-cases are further developed and expressed in more concrete terms they move further right on the continuum.

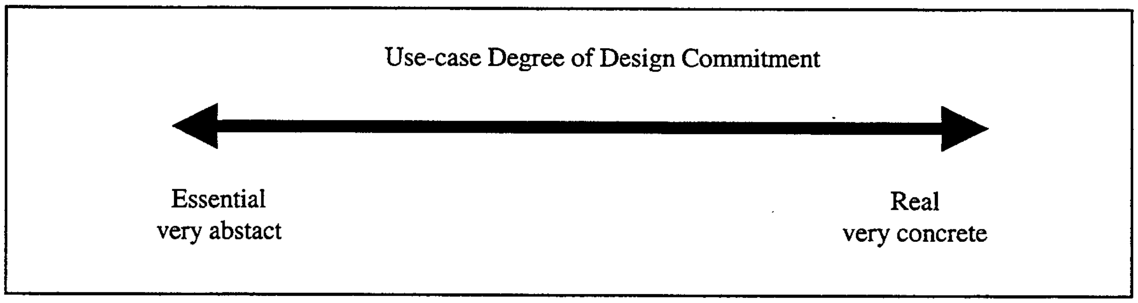


Figure 5.9: Essential and real use-cases exist on a continuum

D. MARKS SYSTEM BEHAVIOR

The UML system-sequence diagram demonstrates the events and actions of actors in MARKS. We chose the system-sequence diagram to show the behavior of MARKS and its subsystems, by providing a description of *what* the system does. It does not, however, provide detail on *how* MARKS, or its subsystems, does its job. The input into the system behavior analysis phase consists of the essential use-cases and conceptual model. The output as shown in Figure 5.10 will be a set of system-sequence diagrams. A system-sequence diagram consists of the concepts involved in the events shown as the labeled boxes at the top of the diagram, the timeline shown as the dotted lines from the concept boxes, and messages passed between the concepts shown as labeled arrows from one concept timeline to another. The messages are numbered to show the sequence of events that will be completed. While the use-case and

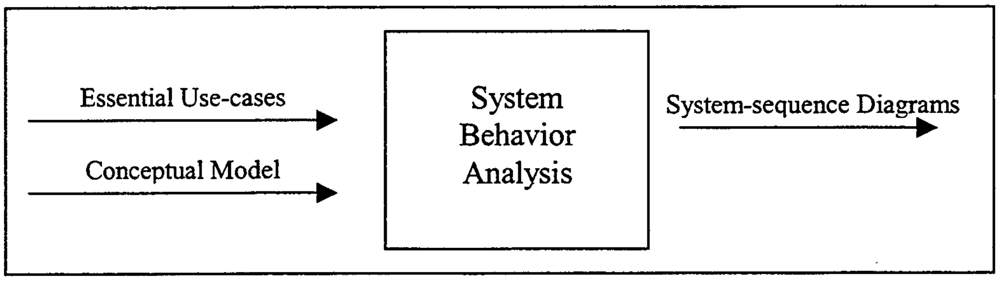


Figure 5.10: System Behavior Analysis

system-sequence diagram shown here are a lower-level view of the system, several system-sequence diagrams were developed to show the actions of the system when interacting with actors external to MARKS. Several iterations of developing the conceptual model and follow-on system-sequence diagrams effectively captured the requirements for MARKS.

The image identification system, Image_ID_System, initiates Figure 5.11 by the system event of analyze_type to the Image_Type_ID class. The event analyze_type is an operation of the Image_ID_System concept. The sequence of events then follows and a list of ships with confidence meter readings is passed back to the image identification system.

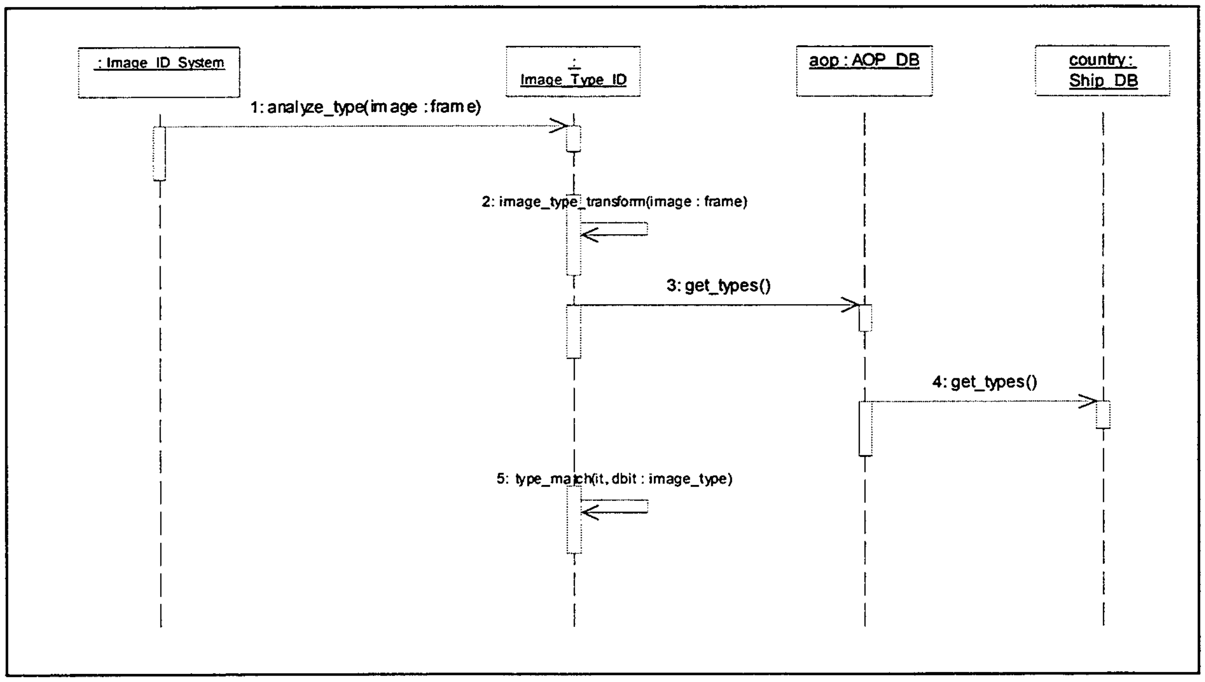

: Image ID System
: Image Type ID
aop : AOP DB
country : Ship DB
1: analyze_type(image : frame)
2: image_type_transform(image : frame)
3: get_types()
4: get_types()
5: type_match(it, dbit : image_type)

Figure 5.11: System-sequence Diagram for Analyze Image Type use-case

The analysis phase for MARKS needed several cycles to refine the requirements. Initially, MARKS was viewed as a black box with only external operators acting on it. Later we "opened up" the black box to develop the low level requirements. The final results of the analysis phase are shown in Appendices H-J.

VI. COMPARISON OF SPEC AND UML MODELS FOR MARKS

We conducted the requirements analysis phase of MARKS using both the SPEC and UML methods described above. The goal of the analysis was not only to develop and specify the requirements but it was to determine which method is better suited for developing mission critical combat systems. It is difficult to make a direct comparison between SPEC and UML. However, as Table 2-1 shows, there is some relation between these two methods of modeling software systems. Table 6-1 gives some simple statistics on the number of elements required to develop MARKS under these two methods. We consider a system goal: *Emitter analysis must correctly identify a maritime contact and give a confidence reading.*

SPEC	Number of Diagrams	Number of Modules
Environment Model	7	16
Functional Specification	19	13
Abstract Architectural Design	2	13

(a)

UML	Total Number
Use-Case Diagrams	3
Use-Case Descriptions	21
Conceptual Model Diagrams	1
Additional Model Diagrams	5
System-Sequence Diagrams	13

(b)

Table 6-1: (a) The number of elements in the model for SPEC and
(b) The number of elements in the model for UML

A. A SPEC FUNCTION

In SPEC we modeled the emitter analyzer as a function shown in Figure 6.1. This function highlights SPEC's reliance on logic to model a system. The function emitter_analyzer has one message called analyze_emitters. When the function is instantiated it declares what area of operations that the system is working in. The message analyze_emitters requires as input a list of emitters and replies back to the caller with a list of ships with associated confidence values. The "heart" of the function is in the logic in the WHERE section. The ultimate goal in using a formal method like SPEC is to provide a clear description of what a software system is supposed to do. The formal specification developed by SPEC could be used to automatically generate the code necessary for the system [22]. Furthermore, an automated

```
FUNCTION emitter_analyzer{aop : area_of_operation}

  INHERIT track
  INHERIT aop_database{aop}
  INHERIT emitter_system_environment
  INHERIT confidence_meter_definitions

  MESSAGE analyze_emitters(el : emitter_list)
    REPLY (s : set{ship_emitter_info})
    WHERE ALL(dps : set{database_pair} :: dps = get_aop(aop) =>
         ALL(x : database_pair :: x IN dps =>
           SOME(sei : ship_emitter_info :: emitter_list_match(el,x) =>
                    sei.s = x.s, sei.cm = ship_confidence(el, x) & sei IN s)))
   --s1 will contain the set of all ships have emitters that match the given emitter list and a confidence
  -- meter associated with how well the lists match.

  CONCEPT ship_emitter_info : type
                 WHERE ship_emitter_info = tuple {s :: ship, cm :: confidence_meter)

  CONCEPT emitter_list_match(el : emitter_list, dp : database_pair)   VALUE (b : boolean)
  --b if emitters in the emitter list match the emitters a ship would would emit

  CONCEPT ship_confidence(el : emitter_list, dp : database_pair)   VALUE (cm : confidence_meter)
  --cm is between 0.0 and 1.0 depending upon match of emitters in list to possible emitters on ship

END
```

Figure 6.1: SPEC function emitter_analyzer

theorem prover could be used to validate the *correctness* of the system produced and verify it against the requirements that were developed for the system. For MARKS, use of SPEC produced a coherent and unambiguous description of system operation, the highlight of which is the well-formulated interactions and interfaces between system components. This proved true when working with Ms. Hermann on her image analysis work: as the programmer for the image classifying system, she felt that SPEC provided a better description of system operation than that of UML. However, the specification produced could be hard to understand for those not familiar with simple forms of predicate calculus seen in the message description for Figure 6.1.

B. A UML SYSTEM-SEQUENCE DIAGRAM

The system-sequence diagram presented in Figure 6.2 addresses the same goal presented in Figure 6.1. The concepts involved with this diagram are MARKS, an Emitter_ID_System, a specific AOP_DB, AOP, and a specific Ship_DB (country). MARKS starts by calling the operation analyze_emitters of the

Emitter_ID_System. The Emitter_ID_System then gets the ship emitters from the specified AOP_DB and the Ship_DB to compare against. The Emitter_ID_System then compares the given emitter list to the ship emitter list it has acquired from the database. The development of this system-sequence diagram proved quite simple and easy to understand for the customer. In general, developing MARKS using UML proved to be quite easy. We used Rational Rose's current Rational Rose 2000 Enterprise Edition to produce the model for the project. Rational Rose was a great aid to production of UML as it allowed for quick development of the system. The customer response to the diagrams developed was overwhelming positive and progress could be readily tracked using this method.

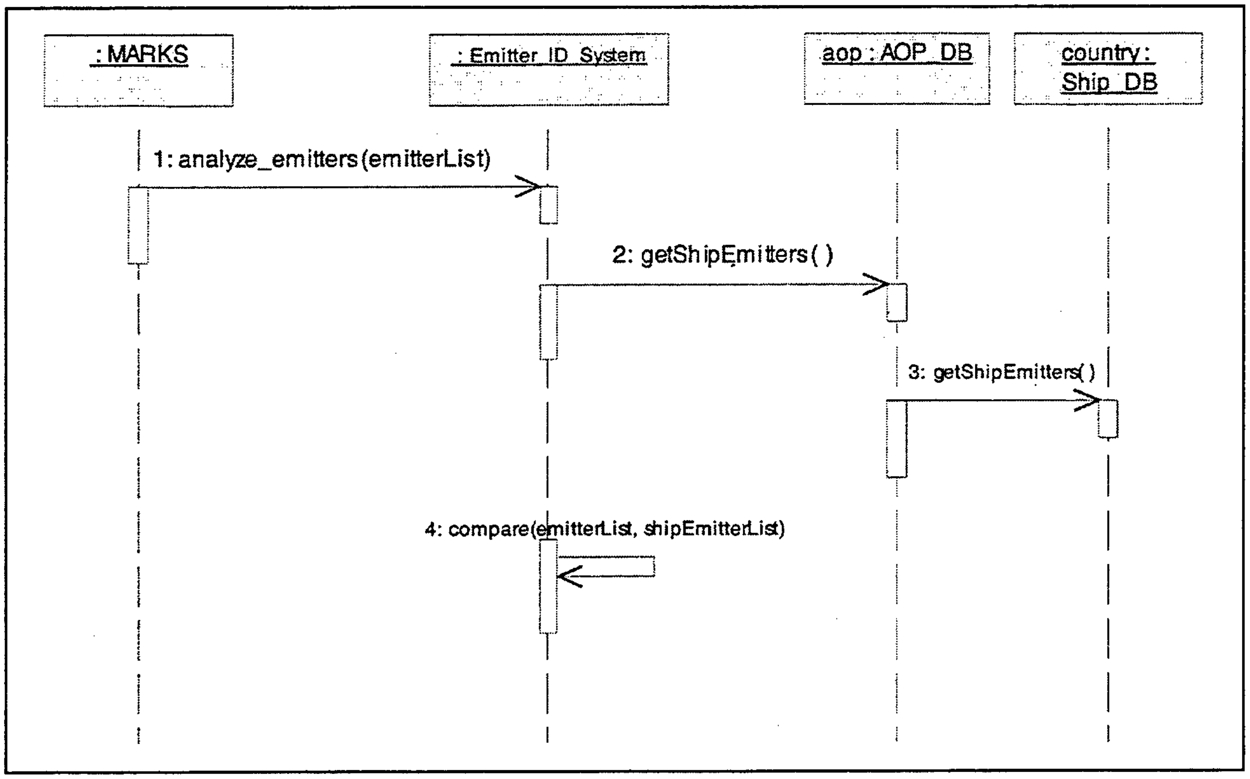


Figure 6.2: UML system-sequence diagram for analyzing emitters

C. RESULTS

While SPEC proved to be a more descriptive model of MARKS, UML was well liked by the customer. With UML it was easier for the customer to understand the requirements of the system and ensure that they matched what their requirements were. However, the informal nature of UML can lead to errors in requirements analysis. The method of development presented in Figures 5.1, 5.3, 5.5, and 5.9 is not a standard under UML. We chose to use this sequence of development to aid in our modeling of MARKS. The method presented for development using SPEC is the standard required. This points out a key difference between a formal and semi-formal method: a formal method follows a prescribed course of development while a semi-formal method does not.

Therefore, the ad-hoc method of developing a system in UML, while giving the developer flexibility in designing the software system, tends to hide errors. The problem with developing software in an object-oriented, semi-formal method is that problems in the analysis and design model may not be uncovered until implementation of the design. While this is true for any method of software development, the non-standardization of UML modeling leads to errors in design. Late-found errors can significantly increase the cost of the project. While Rational Rose can generate code based upon the design, a theorem prover to verify the design's correctness is not available, and only a skeletal implementation of the design is created from the automatic code generation [23].

While UML proved to be excellent for modeling of objects, classes, and simple interactions between them, its lack of formality showed a weakness in modeling of combat systems. The need for clear, concise, and unambiguous specifications highlighted the current failings in semi-formal methods. Semi-formal methods such as UML have proven their worth in developing commercial software but have not in the arena of mission critical combat systems.

Formal methods could be very useful in software development. However, there are many obstacles. The notation of the specification language tends to produce a barrier between the customer, software developer, and programmer. The lack of proper tools to implement a formal method not only reduces the developer's ability to communicate with the customer and the programmers but increases the time required to develop the system. Finally, many commercial companies and the DOD may not be ready to advance the "radical change" of moving to a formal method. The barriers to this include cost and training [23]. While formality in software design may become the norm in the future for software engineers, it is not widely used by the software industry [24]. Although by no means a relief for all problems in software engineering, formal methods are powerful tools that need to be better understood by developers [25].

If, as seems to be the case, UML is becoming the industry standard for modeling of software systems, perhaps we could formalize the process of developing software in UML; many attempts have been made [23][26][27]. A lightweight formal method could be developed using this object-oriented approach. The research performed in this area is promising; generally, its goal is to provide UML with a formal basis while maintaining the look and feel of developing under an object-oriented process.

The Object Constraint Language Specification (OCL) may prove to be an alternative formal language. OCL is being developed for business models [27]. For development of combat systems for DOD, a more rigorous method will be needed. Favre's GSBL and Jia's use of Z both are steps in the right direction. Both of these, however, formalize object classes. Interactions between objects, critical in combat systems, must be formalized before reliance on a formalized UML can take place.

THIS PAGE IS INTENTIONALLY LEFT BLANK

VII. THE FORMAL SPECIFICATION OF MARKS IN SPEC

To continue our design we also developed the abstract architectural design for MARKS. We broke MARKS up into a set of small modules to give a lower-level description of MARKS system behavior. Figure 7.1 shows the modules developed and their dependencies to one another.

A. THE CORE OF MARKS

The modules overall_marks_system, overall_analyzer, ship_list, and overall_operator_interface represent the core of MARKS. These interface with submodules. The machine described in Figure 7.2 represents the method for system initiation. The message start_marks will send two messages to ship_list. The sample Ada code that was derived from this is shown in Figure 7.3. Figure 7.4 shows the machine ship_list which is a state machine which maps a track number to a track; a track represents a ship logged in MARKS. The update messages, update_ship_emitters and update_list, are used to initialize the ship_list.

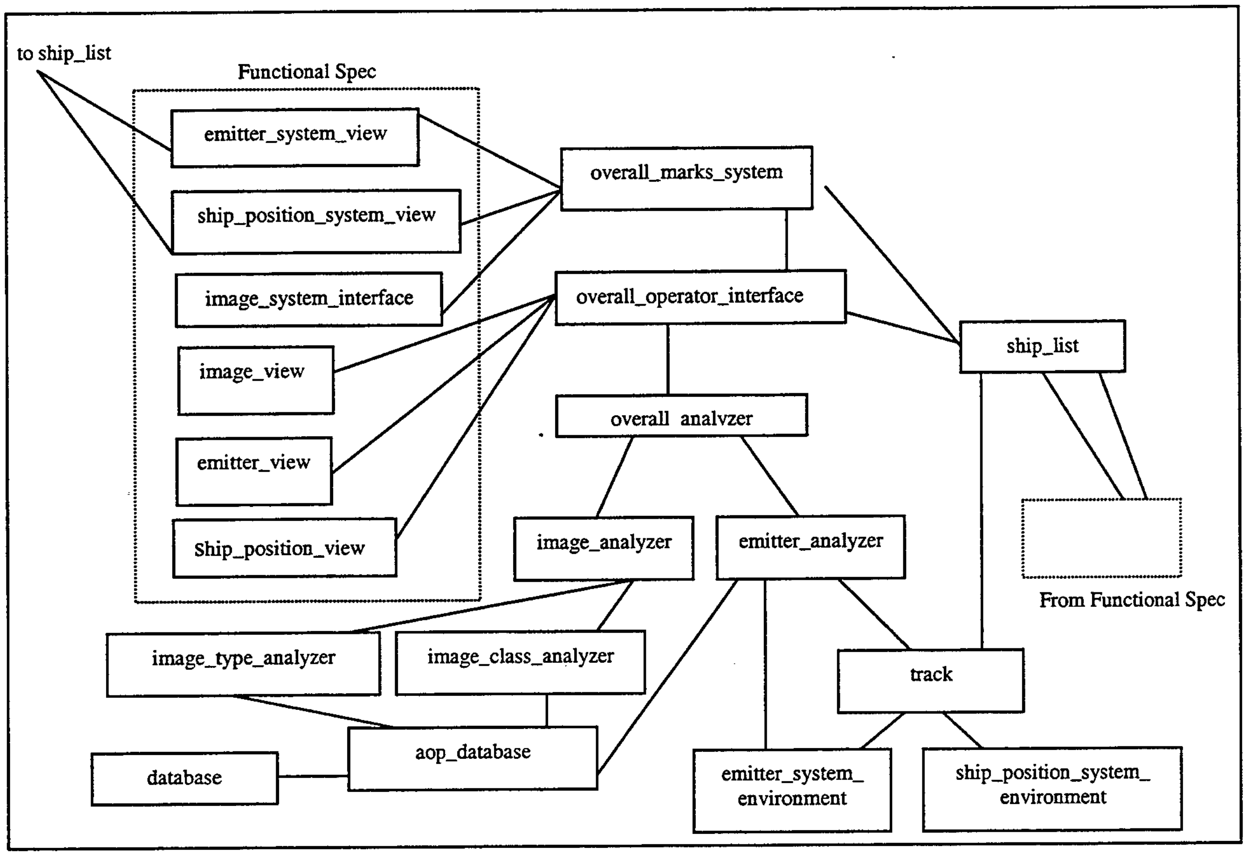


Figure 7.1: Architectural Design Module Dependencies
Note: The dashed box labeled "From Functional Spec" connects to the "to ship_list" lines on the left side.

```
MACHINE overall_marks_system

 INHERIT overall_operator_interface
 INHERIT ship_list
 INHERIT emitter_system_view
 INHERIT image_system_interface
 INHERIT ship_position_system_view

 MESSAGE start_marks
   SEND update_list TO ship_list
   SEND update_ship_emitters TO ship_list
     --update_list and update_ship_emitters are called to initiate updating those
    --systems.  With the inherit   overall_operator_interface, image_system_interface
    -- emitter_system_view, and ship_position_system_view are initialized (i.e. turned on)
END
```

Figure 7.2: Machine for overall_marks_system

```
--overall_marks_system spec
with overall_operator_interface;
with ship_list;
with emitter_system_view;
with image_system_interface;
with ship_position_system_view;

package overall_marks_system is
   procedure start_marks;
end overall_marks_system;

--overall_marks_system body
with overall_operator_interface; use overall_operator_interface;
with ship_list; use ship_list;
with emitter_system_view; use emitter_system_view;
with image_system_interface; use image_system_interface;
with ship_position_system_view; use ship_position_system_view;

package body overall_marks_system is
    procedure start_marks is
        begin
            ship_list.update_list;
            ship_list.update_ship_emitters;
  end start  marks;
```

Figure 7.3: Sample Ada code for overall_marks_system

The core analyzer of MARKS is the overall_analyzer. Figure 7.5 shows this module as a machine with a state comprised of three sets of ship identification, labeled "s"; emitter identification, labeled "em"; and image identification, labeled "im". These sets will change based upon the results of analysis completed on system data. This is accomplished when start_analyzer message is received. The transitions of the

```
MACHINE ship_list

  INHERIT map
  INHERIT ship_position_system_environment
  INHERIT track

  STATE(m: map{track_number, track})
  INVARIANT true
  INITIALLY m = []

  MESSAGE get_symb_info REPLY(symb_map : map{track_number, symbol_info} )
    WHERE ALL(t : track_number :: t IN m => bind(t, get_symbol_info(t), symb_map))
         -- returns a map{track_number, symbol_info} of all the tracks in m

  MESSAGE get_track_info(t_num : track_number) WHEN t_num IN m  REPLY(t : track)
     WHERE t = m[t_num]
      OTHERWISE REPLY EXCEPTION no_such_track  -- returns track info on specified track

  MESSAGE update_list(spt : ship_position_type)
    WHEN spt.tn IN m  TRANSITION m = bind(spt.tn, update(spt),*m[spt.tn])
         -- updates a particular track if the track already exists, update it
      OTHERWISE TRANSITION m = bind( spt.tn, update(spt), *m)
         -- if the track doesn't exist yet, convert the ship_position_type to a track
         --and then add it to the list

  MESSAGE update_ship_emitters(emitter : emitter_data_type, position : coordinate)
    REPLY (m : map{track_number, track}) WHERE ALL(t: track_number :: t IN m)
END
```

Figure 7.4: Machine for ship_list

```
MACHINE overall_analyzer

  INHERIT analyzer_definitions
  INHERIT confidence_meter_definition
  INHERIT emitter_analyzer
  INHERIT image_analyzer

  STATE (s im em : set{ship_id_tuple})
  INVARIANT true
  INITIALLY s = [ ]

  MESSAGE start_analyzer(image : frame, t : track)  REPLY s
    TRANSITION im = analyze_image(image),  em = analyzer_emitters(t.corr_emitters),
           s = correlate_em_im(im,em)


  CONCEPT correlate_em_im(im em: set{ship_id_tuple})  VALUE (s1 : set{ship_id_tuple})
  --Correlates the entries in the im set and em set and returns a new set of ship_id_tuple

END
```

Figure 7.5: Machine for overall_analyzer

emitter and image identification sets occur from interfacing with the image_analyzer and emitter_analyzer as shown in Figure 7.1. The final ship identification set is determined by correlation of these two sets. This is accomplished by comparing the two sets and finding matching ship identifications.

The user interface is represented as a machine called overall_operator_interface (Figure 7.6). The machine maps a track number to the information for a symbol, an image, a track, the display mode, and a

```
MACHINE overall_operator_interface

  INHERIT image_view
  INHERIT emitter_view
  INHERIT ship_position_view
  INHERIT setup_view
  INHERIT maritime_analysis_view
  INHERIT ship_list
  INHERIT time_unit
  INHERIT overall_analyzer

  STATE (m : map{track_number, symbol_info}, image : frame, t : track,
            dm : display_mode, id : set{ship_id})
  INVARIANT true
  INITIALLY get_ship_positions, dm = "setup", image = !, t = !, id = {}

  MESSAGE get_ship_positions SEND update_display(m) TO overall_operator_interface
    SEND get_ship_positions TO overall_operator_interface
    WHERE DELAY = (1 s) TRANSITION m = get_symb_info
   --getting new ship position information and displaying it every 1 s

  MESSAGE get_track(t_num : track_number) WHEN dm = "ships"
      SEND display_track_info(t) TO overall_operator_interface
      TRANSITION t = get_track_info(t_num)
      OTHERWISE REPLY EXCEPTION display_mode_error
  --Displays track information on selected track

  MESSAGE display_emitters(t : track, b : base_location, lob : line_of_bearing) WHEN dm = "ships"
       REPLY done WHERE update_emitters(get_emitters(b,lob))
       OTHERWISE REPLY EXCEPTION display_mode_error
  --Displays emitters for a given track

  MESSAGE display_track_info(t : track) REPLY done WHEN display_mode = "ships" REPLY done
     OTHERWISE REPLY EXCEPTION display_mode_error
  --displays track information given on the screen

  MESSAGE update_display(m1 : map{track_number, symbol_info}) REPLY done
  --updates display of ship positions

  MESSAGE update_emitters(el : emitter_location) REPLY done   -
   -updates emitters displayed on screen
```

Figure 7.6: Machine for overall_operator_interface

```
  MESSAGE get_image(fmi : full_motion_image, f : frame) REPLY done WHEN dm = "image"
      TRANSITION image = capture_image(fmi,f)
       OTHERWISE REPLY EXCEPTION display_mode_error
  --Captures displayed image to memory


MESSAGE stop_image(fmi : full_motion_image) REPLY done  WHEN dm = "image"  WHERE
freeze_image(fmi)
    OTHERWISE REPLY EXCEPTION display_mode_error
  --Freezes a full motion image on 1 frame

MESSAGE start_image(fmi : full_motion_image) WHEN dm = "image" REPLY done
   WHERE unfreeze_image
    OTHERWISE REPLY EXCEPTION display_mode_error
  --Start image back to full motion

MESSAGE set_aop(aop : area_of_operations)  WHEN dm = "setup" REPLY done
   WHERE enter_area_of_operations(aop)
   OTHERWISE REPLY EXCEPTION display_mode_error

MESSAGE set_high_value_unit(hvu : ship) WHEN dm = "setup" REPLY done
   WHERE enter_high_value_unit(hvu)
    OTHERWISE REPLY EXCEPTION display_mode_error

 MESSAGE set_home_station(hs : ship) WHEN dm = "setup" REPLY done
    WHERE enter_home_station(hs)
     OTHERWISE REPLY EXCEPTION display_mode_error

MESSAGE analyze_data  SEND display_analyzer_results TO overall_operator_interface
    TRANSITION dm = "analyzer", id = start_analyzer(image, t)

MESSAGE display_analyzer_results WHEN dm = "analzyer" REPLY done
      OTHERWISE REPLY EXCEPTION display_mode_error
   --displays results of analyzing data stored in id

MESSAGE display_ship_positions REPLY done TRANSITION display_mode = "ships"
  --Switches display to ship position mode
MESSAGE display_image REPLY done TRANSITION display_mode = "image"
 --Switches display to image mode

MESSAGE display_setup REPLY done TRANSITION display_mode = "setup"
  --Switches display to setup mode

CONCEPT display_mode : set{string}
  WHERE display_mode = {"ships", "image", "setup", "analyzer"}

END
```

Figure 7.6: continued: Machine for overall_operator_interface

ship identification. When this machine is initialized the user is taken through system setup. The overall_operator_interface lets the user use subsequent views outlined in the functional specification as shown in Figure 7.6.

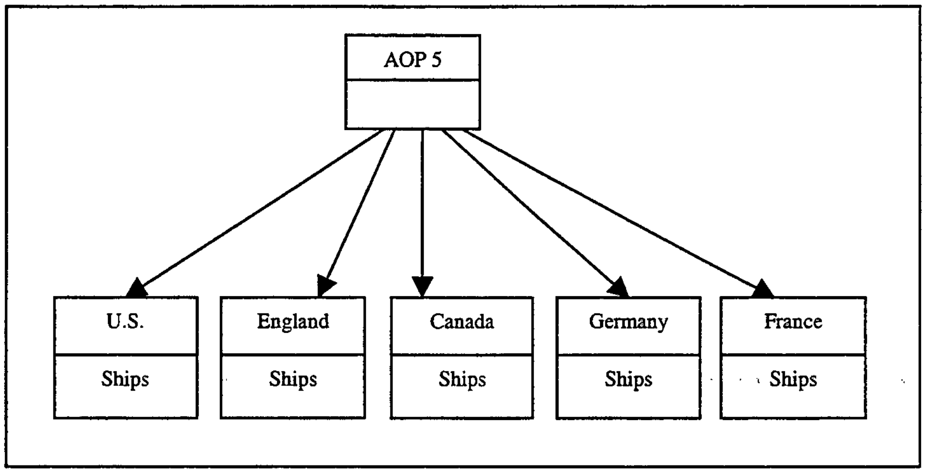


Figure 7.7: Example Database Configuration

B. THE DATABASE

We also modeled the databases. There are ten area of operations (AOP) databases. Each represents a geographical operating area, and would contain a list of ships that operate in the specified AOP. Country databases contain a list of ships that operate out of that country (Figure 7.7). The AOP databases are modeled as state machines, with pairs linking a ship identification and the corresponding database information. The AOP database machine is shown in Figure 7.8.

C. THE MODEL OF MARKS

The complete abstract model of MARKS also includes a goal/requirements tree, an environmental model, a functional specification, and an architectural design. With this, we have taken an imprecise customer-needs statement and modeled it in an unambiguous manner. By no means is this a one-pass method; it took several development loops to flesh out and concisely specify MARKS' requirements, constraints, and actions. The precision required for a mission-critical combat system has been aided by use of a formal development method.

```
MACHINE aop_database{aop : area_of_operation}

  INHERIT position_definitions
  INHERIT ship_definitions
  INHERIT database{ship, database_information}

  STATE(e : set{database_pair}, open : boolean)
  INVARIANT true
  INITIALLY e = load_aop(aop)

  MESSAGE open_aop(aop : area_of_operation)  WHEN ALL(dbn : db ::
     get_db(dbn) IN e => open_db(dbn))
      TRANSITION e.open  OTHERWISE REPLY EXCEPTION aop_error

  MESSAGE close_aop(aop : area_of_operation) WHEN ALL(dbn : db :: get_db(dbn) IN
       e => close_db(dbn))
      TRANSITION ~e.open  OTHERWISE REPLY EXCEPTION aop_error

  MESSAGE get_aop(aop : area_of_operation)   REPLY e

  MESSAGE load_aop(aop : area_of_operation)
    WHEN ALL(dbn : database :: db_in_aop(dbn,aop) => ALL(s : ship, adm : aop_db_map ::
            adm = get_db(dbn) & s IN adm => ALL(di : database_information,   dp : database_pair ::
           dp.s = s & dp.di = adm.m[s])))
      TRANSITION e = *e U dp  OTHERWISE  REPLY EXCEPTION aop_error

  CONCEPT database_information : type  WHERE database_information = tuple(s :: ship,
        sc :: ship_class, st :: ship_type, eml :: emitter_list, gl :: graphical_location, it :: image_type,
        ic :: image_class)
        --Need to define emmiter_list, graphical_location, and analyzed_image

  CONCEPT database_pair : type WHERE database_pair = tuple{s :: ship, di :: database_information}

  CONCEPT aop_db_map : type WHERE aop_db_map = map{ship, database_information}

  CONCEPT db_in_aop(db : database, aop : area_of_operation) VALUE (b : boolean)
    ---b if the country operates in that aop
END
```

Figure 7.8: Machine for aop_database

THIS PAGE IS INTENTIONALLY LEFT BLANK

VIII. CONCLUSIONS

This study suggests that the needs of the Navy lie not only in developing combat systems that meet the needs of the user at sea, but in developing these systems in a timely and accurate manner. The current semi-formal and informal methods of developing software do not meet the standards required for mission-critical combat systems. Such systems require high fault tolerance which cannot be provided by use of these methods. While a formal method initially may not be as satisfying to the customer, the benefits of detecting errors in design early weigh heavily its favor. But many software developers will not commit to such a process. For widespread use of formal methods, a more feasible method of implementing them must arise. Lightweight formal methods may allow the developer to partially develop a software system and test only that portions; successful applications of lightweight formal methods have been shown in case studies [28].

A. FURTHER STUDY

The development of MARKS needs to continue. Follow-on research should define the concrete architectural design and develop the user interface for the customer. The requirements presented here need further development to determine not only their feasibility but the usefulness to the customer. Further research needs to be done to determine whether the system can accurately synthesize the data from the image system, emitter system, and ship positioning system. While initial research has suggested that a small set of ships can be distinguished by their images, the ability to do this for a larger database has not been tested. Obstacles to developing in a formal environment have been outlined above and should be addressed. The need for automated tool support is evident. A formal notational base for UML needs to be investigated due to the need for highly fault-tolerant software systems in the U.S. military.

THIS PAGE IS INTENTIONALLY LEFT BLANK

APPENDIX A. MARKS GOAL/REQUIREMENTS TREE

G.1. The goal of the Maritime Analyzer Recognition Knowledge System (MARKS) is to identify maritime contacts.

- G1.1 MARKS should provide a cumulative analysis based on image, known ship position, detected emitters and likelihood of a particular type ship based on known operating areas.
  - G1.1.1: MARKS should incorporate an image analysis system into its formula for identifying maritime contacts.
    - R1.1.1.1: An imaging system should be a part of the station using MARKS.
    - G1.1.1.2: Image analysis must correctly identify a maritime contact giving a confidence reading.
      - R1.1.1.2.1: An image capture capability should be incorporated in MARKS.
      - G1.1.1.2.2: System should correlate ship type and class with image attributes to identify ship.
        - G1.1.1.2.2.1: System should identify type of ship within a specified amount of time.
          - R1.1.1.2.2.1.1 A list of ship types that match the image should be provided
          - G1.1.1.2.2.2: System should identify class of ship within a specified amount of time.
            - R1.1.1.2.2.2.1 A list of ship classes that match the image should be provided
  - G1.1.2: MARKS should incorporate an emitter analysis system into its formula for identifying maritime contacts
    - R1.1.2.1: An emitter system should be a part of the station using MARKS.
    - G1.1.2.2: Emitter analysis must correctly identify a maritime contact giving a confidence reading.
      - R1.1.2.2.1: MARKS should have an interface to the emitter system.
      - R1.1.2.2.2: The interface should provide a line of bearing for the emitter system to concentrate on.
      - R1.1.2.2.3: The emitter system should provide a set of emitters detected down the specified line of bearing.
    - R1.1.2.3: MARKS should analyze detected emitters periodically to find contacts of interest.
  - G1.1.3: MARKS should incorporate a ship-positioning analysis system into its formula for identifying maritime contacts.
    - R1.1.3.1: A ship position system should be a part of the station using MARKS.
    - G1.1.3.2: Ship position analysis should provide a list of ships located near the given coordinate.
      - R1.1.3.2.1: MARKS should have an interface to the ship position system.
      - R1.1.3.2.2: The interface should provide a coordinate to the ship position system.
    - R1.1.3.3: MARKS should analyze detected ships periodically to find contacts of interest.

- R1.2: MARKS should provide a interface to the operator.
  - G1.2.1: A graphical user interface should be utilized to convey information to the operator.
    - R1.2.1.1: Real time image of maritime contact should be displayed.
    - R1.2.1.2: Detected emitter list down a line of bearing should be displayed.
    - R1.2.1.3: Known ship positions around a specified coordinate should be displayed.
    - R1.2.1.4: Results of analysis with confidence meter reading should be displayed.
  - G1.2.2 A graphical user interface should be utilized to get information from the operator.
    - G1.2.2.1: MARKS should provide a method for capturing a still image of displayed maritime image.
      - R1.2.2.1.1 MARKS should allow operator to freeze/unfreeze displayed full motion image
      - R1.2.2.1.2 MARKS should allow operator to capture a frozen image
    - G1.2.2.2: MARKS should provide a method for intializing emitter search.
      - R1.2.2.2.1: MARKS should provide a method for entering the line of bearing and the point from which that bearing eminates.
    - G1.2.2.3: MARKS should provide a method for intializng ship position search.
      - R1.2.2.3.1: MARKS should provide a method for entering a coordinate and a range from that coordinate to concentrate ship position analysis.
    - R1.2.2.4: A method to initialize maritime contact analyzing.

G1.2.2.5: MARKS should provide a setup display for the operator that contains area of operations, high value unit, and home station

- R1.2.2.5.1: MARKS should provide a method for entering the area of operations
- R1.2.2.5.2: MARKS should provide a method for entering the high value unit
- R1.2.2.5.3: MARKS should provide a method for entering the home station

APPENDIX B. MARKS ENVIRONMENT MODEL (SPEC)

```
DEFINITION marks_definitions

  INHERIT station_definitions
  INHERIT sensor_definitions
  INHERIT image_system_definitions
  INHERIT emitter_analysis_system_definitions
  INHERIT ship_position_analysis_definitions
  INHERIT operator_definitions

  EXPORT marks

  CONCEPT  marks: software_system
    WHERE proposed(marks)              --the goal of the system is to build MARKS
    & uses(operator, marks)             --a human operator uses MARKS
    & uses(marks, image_analysis_system)
    & uses(marks, emitter_analysis_system)
    & uses(marks, ship_position_analysis)

END

-----------------------------------------------------------------------------------------------------------
--A station must have a ship position system, emitter system, and an image system
-----------------------------------------------------------------------------------------------------------
DEFINITION station_definitions
  INHERIT system
  INHERIT sensor_definitions
  INHERIT relationship2

  CONCEPT station: type WHERE ALL(st : station :: part_of(ship_position_sytem, station)
     & part_of(emmiter_system, station) & part_of(image_analysis_system, station))
    --station will have ship_position_system, emitter_system, image_system

END

-----------------------------------------------------------------------------------------------------------
--The actual hardware systems that make are utilized by MARKS are defined here
-----------------------------------------------------------------------------------------------------------
DEFINITION sensor_definitions
  CONCEPT ship_position_system : hardware_system
  CONCEPT emitter_system: hardware_system
  CONCEPT image_system: hardware_system
END
```

```
------------------------------------------------------------------------------------------------------------
--The Image Analysis System is a proposed software system that uses the image system, has an image
--capture capability, and has image analyzer capability with type and class analyzers
------------------------------------------------------------------------------------------------------------
DEFINITION image_analysis_system_definitions

  INHERIT system
  INHERIT confidence_meter_definitions
  INHERIT image_type_analyzer_definitions
  INHERIT image_class_analyzer_definitions
  IMPORT Subtype FROM type

  CONCEPT image_analysis_system : software_system WHERE proposed(image_analysis_system)
        & uses(image_analysis_system, image_system) & image_capture_capability
        & have_image_analyzer
  --The image analysis system is a proposed softare system that uses the image system

  CONCEPT class_type_match(st : ship_type, sc : ship_class) VALUE (b : boolean)
  --b if ship type, st, correlates with ship class, sc, then

  CONCEPT have_image_analyzer : boolean
    WHERE have_image_analyzer <=> have_image_type_analyzer & have_image_class_analyzer,
        goal(have_image_analyzer, image_analysis_system)
END

------------------------------------------------------------------------------------------------------------
--Specifies what an image type analyzer should return
------------------------------------------------------------------------------------------------------------
DEFINITION image_type_analyzer_definitions

  INHERIT ship_definitions
  INHERIT image_capture_definitions

  CONCEPT image_type_analyzer(image : frame) VALUE (st : ship_type)
    WHERE SOME(s : ship_type :: ship_type_match(image, st) => st = s)
      --st is ship type returned that the image correlates

  CONCEPT have_image_type_analyzer : boolean
    WHERE have_image_type_analyzer <=> SOME(sh : ship ::
      SOME(st : ship_type :: st = image_type_analyzer(sh))),
        goal(have_image_type_analyzer, image_analysis_system)

  CONCEPT ship_type_match(image : frame, st : ship_type) VALUE (b : boolean)
    --b if frame, image, correlates to a ship type, st

  CONCEPT image_type : type
END
```

```
------------------------------------------------------------------------------------------------------------
--Specifies what an image class analyzer should return
------------------------------------------------------------------------------------------------------------
DEFINITION image_class_analyzer_definitions

  INHERIT ship_definitions
  INHERIT image_capture_definitions

  CONCEPT image_class_analyzer(image : frame) VALUE (sc : ship_class)
   WHERE ship_class_match(image,sc)
  --sc is ship class returned that the frame, image, correlates

  CONCEPT have_image_class_analyzer : boolean
    WHERE have_image_class_analyzer <=> SOME(sh : ship ::
      SOME(sc : ship_class :: sc = image_class_analyzer(sh)))

  CONCEPT ship_class_match(image : frame, sc : ship_class) VALUE (b : boolean)
  --b if frame, image, correlates to a ship class, sc

  CONCEPT image_class : type

END

------------------------------------------------------------------------------------------------------------
-- MARKS is being developed to identify maritime contacts (ships). Ships are broken down into type and
-- class.
------------------------------------------------------------------------------------------------------------
DEFINITION ship_definitions

  CONCEPT ship : type

  CONCEPT ship_type : type   WHERE Subtype(ship_type, ship)

  CONCEPT ship_class : type WHERE Subtype(ship_class, ship)

  CONCEPT valid_ship(s : ship) VALUE (b : boolean)    --b if given ship is a valid ship

END

------------------------------------------------------------------------------------------------------------
--Defines the image capture capability of MARKS
------------------------------------------------------------------------------------------------------------
DEFINITION image_capture_definitions

  INHERIT hardware_concepts2

  CONCEPT frame : type

  CONCEPT full_motion_image : type
    WHERE ALL(fmi : full_motion_image :: SOME(f:frame :: consists_of(fmi,f) &
          frame_rate(fmi) >= 25))  --full motion image is defined as an image with a frame rate
                                   --of 25 frames per second or greater

  CONCEPT frame_rate(fmi : full_motion_image) VALUE (fps : nat)
  --fps (frames per second) is a natural number returned that represents the fps of the full motion image, fmi
  CONCEPT image_capture(fmi : full_motion_image) VALUE (b : boolean)
   WHERE b <=> SOME(f : frame :: consists_of(fmi,f) & stored_in_memory(f))
  --image is captured if the frame f is in the full motion image fmi and f is stored in memory

  CONCEPT image_capture_capability : boolean
    WHERE image_capture_capability <=> ALL(fmi : full_motion_image
         :: display_video(fmi) & image_capture(fmi)),
    goal(image_capture_capability, image_analysis_system)

END

------------------------------------------------------------------------------------------------------------------
--The emitter analysis system is a proposed software system that should correlate known emitters to ships
--utilizing those emitters.
------------------------------------------------------------------------------------------------------------------
DEFINITION emitter_analysis_system_definitions

  INHERIT system
  INHERIT confidence_meter_definitions
  IMPORT Subtype FROM type

  CONCEPT emitter_analysis_system : software_system WHERE proposed(emitter_analysis_system)
        & uses(emitter_analysis_system, emitter_system)
END

------------------------------------------------------------------------------------------------------------------
--The ship position analysis system is a proposed software system that should correlate known ship
--positions to proposed maritime contact.
------------------------------------------------------------------------------------------------------------------
DEFINITION ship_position_analysis_definitions

  INHERIT system
  INHERIT confidence_meter_definitions

  CONCEPT ship_position_analysis : software_system  WHERE proposed(station_position_analysis)
    & uses(station_position_analysis, station_position_system)

END

------------------------------------------------------------------------------------------------------------------
--The confidence meter is a rating of the reliability of the identification being made by MARKS
------------------------------------------------------------------------------------------------------------------
DEFINITION confidence_meter_definitions

  IMPORT Subtype FROM type

  CONCEPT confidence_meter : type WHERE Subtype(confidence_meter , real)
        & ALL(cm :  confidence_meter :: 0.0 <= cm <= 1.0)

END
```

```
------------------------------------------------------------------------------------------------
--Currently only 1 class of user is defined "Operator"
------------------------------------------------------------------------------------------------
DEFINITION operator_definitions

  INHERIT user
  IMPORT marks FROM marks_definitions

  CONCEPT operator : User_class WHERE uses(operator, marks_system)

END


DEFINITION position_definitions

  INHERIT location
  INHERIT time

  IMPORT Subtype FROM type


  CONCEPT line_of_bearing : type WHERE Subtype(line_of_bearing, degree)
    --A line of bearing must be within 0 to 359 degrees

  CONCEPT range : type WHERE Subtype(range, real) &  ALL(r : range :: r > 0)
    --A range is a real number greater than 0

  CONCEPT coordinate : type
   WHERE coordinate = tuple(latitude :: deg_min_sec, longitude :: deg_min_sec)

  CONCEPT deg_min_sec : type
   WHERE deg_min_sec = tuple(deg :: degree, min :: minute, sec :: second)

  CONCEPT degree : type
   WHERE Subtype(degree, number) &  ALL(deg : degree :: 0 <= deg <= 359)
  --degree must be within 0 to 359 degrees

  CONCEPT area_of_operation : type WHERE Subtype(area_of_operation, number) &
         ALL(aop : area_of_operation :: 0 <= aop <= 10)

END
```

```
-----------------------------------------------------------------------------------------------------------------------
--Defining new relationships not defined in the given "relationship"
--definition
-----------------------------------------------------------------------------------------------------------------------
DEFINITION relationship2

  INHERIT system
  INHERIT station_definitions
  INHERIT image_capture_definitions

   CONCEPT part_of(hs : hardware_system, st : station) VALUE(b : boolean)
  --True if some hardware system is a part of some station

   CONCEPT consists_of(fmi : full_motion_image, f : frame)  VALUE (b : boolean)
   --b if full motion image, fmi, consists of frame, f

END

-----------------------------------------------------------------------------------------------------------------------
--Extensions of the hardware concepts defined in SPEC
-----------------------------------------------------------------------------------------------------------------------
DEFINITION hardware_concepts2
  INHERIT processor_concepts --gives "processor" & "main_memory_size" for RAM
  INHERIT memory_concepts    --gives "Mbyte"


   CONCEPT main_memory : type WHERE ALL(p : processor :: main_memory_size(p) >= (256 Mbyte))
   --Need at least 256 megabytes of main memory

   CONCEPT size_of(a : any) VALUE (size : nat)
   --Returns the amount of memory in megabytes that a would take up

   CONCEPT memory_available(m : main_memory) VALUE (avail : nat)
   --Returns the amount of main memory currently available for storage

   CONCEPT stored_in_memory(a : any) VALUE (b : boolean)
      WHERE size_of(a) < memory_available(m)
   -- b if a is stored in main memory if enough main memory is available

END
```

APPENDIX C. MARKS FUNCTIONAL SPECIFICATION (SPEC)

```
MACHINE overall_operator_interface

  INHERIT image_view
  INHERIT emitter_view
  INHERIT ship_position_view
  INHERIT setup_view
  INHERIT maritime_analysis_view
  INHERIT ship_list
  INHERIT time_unit
  INHERIT overall_analyzer

  STATE (m : map{track_number, symbol_info}, image : frame, t : track,  dm : display_mode,
          id : set{ship_id})
  INVARIANT true
  INITIALLY get_ship_positions, dm = "setup", image = !, t = !, id = { }


  MESSAGE get_ship_positions
    SEND update_display(m) TO overall_operator_interface
    SEND get_ship_positions TO overall_operator_interface
        WHERE DELAY = (1 s)  TRANSITION m = get_symb_info
  --getting new ship position information and displaying it every 1 s

-----------------------------------------------------------------------------------------------------------------------
  --"ships" display mode messages
-----------------------------------------------------------------------------------------------------------------------
  MESSAGE get_track(t_num : track_number)
    WHEN dm = "ships"  SEND display_track_info(t) TO overall_operator_interface
      TRANSITION t = get_track_info(t_num) OTHERWISE REPLY EXCEPTION display_mode_error
  --Displays track information on selected track

  MESSAGE display_emitters(t : track, b : base_location, lob : line_of_bearing)
    WHEN dm = "ships" REPLY done WHERE update_emitters(get_emitters(b,lob))
    OTHERWISE REPLY EXCEPTION display_mode_error
  --Displays emitters for a given track

  MESSAGE display_track_info(t : track) REPLY done  WHEN display_mode = "ships"  REPLY done
    OTHERWISE REPLY EXCEPTION display_mode_error
  --displays track information given on the screen


  MESSAGE update_display(m1 : map{track_number, symbol_info}) REPLY done
  --updates display of ship positions
```

```
MESSAGE update_emitters(el : emitter_location) REPLY done
--updates emitters displayed on screen

--------------------------------------------------------------------------------------------------------
--"image" display mode messages
--------------------------------------------------------------------------------------------------------
MESSAGE get_image(fmi : full_motion_image, f : frame) REPLY done
  WHEN dm = "image" TRANSITION image = capture_image(fmi,f)
  OTHERWISE REPLY EXCEPTION display_mode_error
--Captures displayed image to memory

MESSAGE stop_image(fmi : full_motion_image) REPLY done
  WHEN dm = "image" WHERE freeze_image(fmi)
  OTHERWISE REPLY EXCEPTION display_mode_error
--Freezes a full motion image on 1 frame

MESSAGE start_image(fmi : full_motion_image) REPLY done
  WHEN dm = "image"  WHERE unfreeze_image
  OTHERWISE REPLY EXCEPTION display_mode_error
--Start image back to full motion

--------------------------------------------------------------------------------------------------------
--"setup" display mode messages
--------------------------------------------------------------------------------------------------------
 MESSAGE set_aop(aop : area_of_operations)
    WHEN dm = "setup" REPLY done  WHERE enter_area_of_operations(aop)
    OTHERWISE REPLY EXCEPTION display_mode_error

 MESSAGE set_high_value_unit(hvu : ship)
    WHEN dm = "setup" REPLY done WHERE enter_high_value_unit(hvu)
    OTHERWISE REPLY EXCEPTION display_mode_error

 MESSAGE set_home_station(hs : ship)
    WHEN dm = "setup" REPLY done WHERE enter_home_station(hs)
    OTHERWISE REPLY EXCEPTION display_mode_error

--------------------------------------------------------------------------------------------------------
--"analyzer" dislay mode messages
--------------------------------------------------------------------------------------------------------
  MESSAGE analyze_data
   SEND display_analyzer_results TO overall_operator_interface
   TRANSITION dm = "analyzer", id = start_analyzer(image, t)


  MESSAGE display_analyzer_results
    WHEN dm = "analzyer" REPLY done  OTHERWISE REPLY EXCEPTION display_mode_error
  --displays results of analyzing data stored in id


--------------------------------------------------------------------------------------------------------
--Toggling between display modes
--------------------------------------------------------------------------------------------------------

MESSAGE display_ship_positions REPLY done TRANSITION display_mode = "ships"
--Switches display to ship position mode
```

```
  MESSAGE display_image REPLY done  TRANSITION display_mode = "image"
 --Switches display to image mode

  MESSAGE display_setup REPLY done TRANSITION display_mode = "setup"
  --Switches display to setup mode

  CONCEPT display_mode : set{string} WHERE display_mode = {"ships", "image", "setup", "analyzer"}
END

------------------------------------------------------------------------------------------------------------------
-- image_view is the interface that the operator "sees" when
--utilizing the imaging system of MARKS.
------------------------------------------------------------------------------------------------------------------
MACHINE image_view

    INHERIT image_system_interface    --full_motion_image comes from this
    INHERIT image_capture_definitions


    STATE (fmi : full_motion_image)
    INVARIANT true                    --full motion image is always streaming for display
    INITIALLY fmi = update_image      --initial full motion image is current updated video

   MESSAGE freeze_image(fmi : full_motion_image)
     WHEN SOME(f : frame :: f IN *fmi) TRANSITION fmi = freeze_image(*fmi,image(*fmi))
     OTHERWISE REPLY EXCEPTION no_image
    --full motion video consists of 1 frame that is updated 25 times a second...virtually "freezing" that frame
   --for display

   MESSAGE unfreeze_image TRANSITION fmi = update_image
    --Get updated set of frames for display

  MESSAGE capture_image(fmi : full_motion_video, f : frame)
    REPLY (f : frame) WHERE image_frozen(fmi)& f = image_capture(fmi)
    --frame returned is a frame from the frozen full motion image

  Concept image_frozen(fmi : full_motion_image) VALUE (b : boolean)
    WHERE b <=> ALL(f1 f2: frame :: f1 IN fmi & f2 IN fmi :: f1 = f2)
    --All frames in full motion image are same frame

  Concept image_capture(fmi : full_motion_image) VALUE (f : frame)
    --frame returned is current frame displayed
END

------------------------------------------------------------------------------------------------------------------
--This FUNCTION serves the operator interface for entering information
--about emitters.
------------------------------------------------------------------------------------------------------------------
FUNCTION emitter_view

    INHERIT position_definitions
    INHERIT emitter_system_environment
    IMPORT Subtype from type
  MESSAGE get_emitters(lob : line_of_bearing, bl : base_location) REPLY (el : emitter_location)
    WHERE el.lob = lob & el.base = bl
  --The operator will enter a line of bearing and a base location (where the bearing is being received, i.e.
  --homeship). The reply gives an emitter location that combines the bearing and base location into 1 type

  CONCEPT base_location : type WHERE Subtype(base_location, coordinate)
  --base location is the coordinate of the sensor that "senses" an emitter

  CONCEPT emitter_location : type
     WHERE emitter_location = tuple{lob :: line_of_bearing, base :: base_location}
  --The combined datatype of line_of_bearing and base_location
END

------------------------------------------------------------------------------------------------------------
-- This FUNCTION serves the operator interface for entering information about ship positions.
------------------------------------------------------------------------------------------------------------
FUNCTION ship_position_view

  INHERIT position_definitions
  IMPORT Subtype from type

 MESSAGE get_ships(coord : coordinate, r : range) REPLY (c : locate) WHERE c.coord = coord & c.r = r
  --The operator will enter a coordinate and a range around that coordinate to locate ships

  CONCEPT  locate : type    WHERE locate = tuple{coord : coordinate, r : range}

END

------------------------------------------------------------------------------------------------------------
--setup_view acts retains the values of entered by the operator for the current area of operations, what the
--high value unit is (ex. the aircraft carrier in the battle group), and what the home ship is.
------------------------------------------------------------------------------------------------------------
MACHINE setup_view

  INHERIT ship_definitions

  STATE {setup = setup_info}
  INVARIANT valid_data
  INITIALLY setup.aop = 0

  MESSAGE enter_area_of_operations(aop : area_of_operation)  TRANSITION setup.aop = aop

  MESSAGE enter_high_value_unit(hvu : ship) TRANSITION setup.hvu = hvu

  MESSAGE enter_home_station(hs : ship) TRANSITION setup.hs = hs

  CONCEPT setup_info : type  WHERE setup_info = tuple{aop :: area_of_operation, hvu hs :: ship}

  CONCEPT valid_data VALUE (b : boolean)
    WHERE b <=> (1 <= setup.aop <= 10) && valid_ship(setup.hvu) && valid_ship(setup.hs)
  --Any valid ship can be designated as the high value unit and home ship.  The valid operating areas
 --around the world must meet the criteria of 1 to 10.

END
```

```
---------------------------------------------------------------------------------------------------------------
-- ship_postion_system_view acts like a queue ship_postion_system broadcasts
-- ship_postion_system_tracks which are added to the queue MARKS will remove items from the queue for
--its use
---------------------------------------------------------------------------------------------------------------
MACHINE ship_position_system_view
 INHERIT ship_position_system_environment

   STATE (e : sequence{ship_position_type})
   INVARIANT true
   INITIALLY e = [ ]

 MESSAGE broadcasted_ships(ships : sequence{ship_position_type})  TRANSITION e = *e|ships
 --When actual ship position system broadcast ships, these shipsare added to the current list of ships held by
 --is interface

 MESSAGE get_ship  WHEN e ~= [ ] REPLY (ship : ship_posit_type)  TRANSITION e = ship|*e
  OTHERWISE REPLY EXCEPTION no_ship_position_system_updates
END


---------------------------------------------------------------------------------------------------------------
-- defines the ship_position_system_environment
---------------------------------------------------------------------------------------------------------------
DEFINITION ship_position_system_environment

  INHERIT position_definitions

  CONCEPT ship_position_type : type
    -- ship_position_type is a record containing all the relevant information about a track at a particular
    --moment
    WHERE ship_position_type = tuple{tn :: track_number, cat :: category, i :: ship_position_system_id,
                                     t :: tag, coord :: coordinate,  c :: course, s :: speed}
  CONCEPT ship_position_track_number : type

  CONCEPT ship_position(spt : ship_position_type) VALUE (stn : ship_position_track_number)
     WHERE One_to_one(ship_position_type)

  CONCEPT category : type --category can be any of the following values: air, surface, subsurface

  CONCEPT ship_position_system_id : type  -- id can be any of the following values: friend, unknown
                                          --assumed friend, unknown evaluated, unknown assumed
                                          --enemy, hostile
  CONCEPT course : type  WHERE Subtype(course, degree)

  CONCEPT speed : type WHERE Subtype(speed, number) & ALL(s : speed :: s >= 0)

  CONCEPT tag : type

END
```

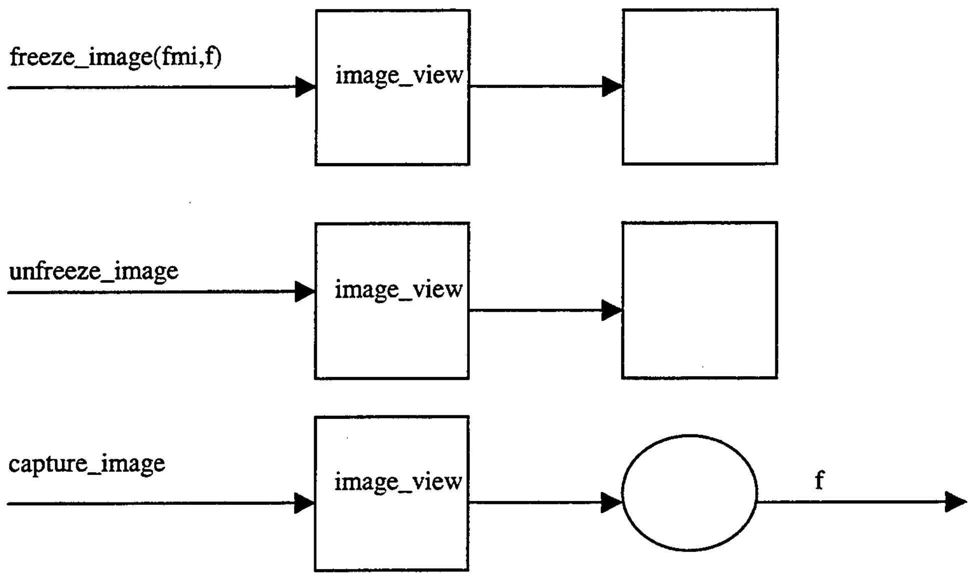
```
----------------------------------------------------------------------------------------------------------------------------
-- emitter_system_view acts like a queue emitter_system broadcasts emitter_tracks which are added to the
--queue MARKS will remove items from the queue for its use
----------------------------------------------------------------------------------------------------------------------------
MACHINE emitter_system_view

  INHERIT emitter_system_environment
  INHERIT position_definitions

   STATE (e : sequence{emitter_data_type})
   INVARIANT true
   INITIALLY e = [ ]

   MESSAGE broadcasted_emitters(emitters : sequence{emitter_data_type})
     TRANSITION e = *e|emitters
   --When actual emitter system broadcast emitters then these emitters are added to the current list of
   --emitters held by this interface

   MESSAGE get_emitter
     WHEN *e ~= [ ] --next emitter is taken from front of sequence and deleted
      TRANSITION *e = emitter|e  REPLY (emitter : emitter_data_type)
    OTHERWISE REPLY EXCEPTION no_emitter_updates
       -- queue underflow

END

----------------------------------------------------------------------------------------------------------------------------
-- emitter_system_environment definitions
----------------------------------------------------------------------------------------------------------------------------
DEFINE emitter_system_environment

  INHERIT position_definitions

  CONCEPT emitter_location : type
    WHERE emitter_location = tuple(bl :: base_location, lob :: line_of_bearing)

 CONCEPT base_location : type WHERE Subtype(base_location,coordinate)

 CONCEPT emitter_system_track_number : type

 CONCEPT emitter : type  -- the name of the emitter

 CONCEPT emitter_data_type : type
    WHERE emitter_data_type = tuple{estn : emitter_system_track_number,
                                                em :: emitter, el : emitter_location}

CONCEPT bearing_accuracy VALUE ba  WHERE ba = 7
       -- the accuracy of the emitter system bearings

END
```

```
FUNCTION image_system

  INHERIT image_capture_definitions

  MESSAGE get_new_image(fmi : full_motion_image)SEND new_image(fmi) to image_system_interface

END

----------------------------------------------------------------------------------------------------
--Full motion images consists of sets of frames displayed at a rate of 25 frames per second.
----------------------------------------------------------------------------------------------------
TYPE full_motion_image

  INHERIT image_capture_definitions

   MODEL(full_motion_image : sequence{frame}) --A video full motion images consists of a set of frames
   INVARIANT   ALL(fmi : full_motion_image :: frame_rate(fmi) >= 25)

   MESSAGE update_image(fmi : full_motion_image)
       REPLY (fmi2 : full_motion_image)  WHERE fmi2 = images(fmi)
    --Given a full motion image display that full motion image

  MESSAGE freeze_image(fmi : full_motion_image, f : frame)
     REPLY(fmi2 : full_motion_image)  WHERE fmi2 = {f}
     --Given the current full motion image and a specific frame, just display that frame

  --Supporting Definitions
  CONCEPT images(fmi : full_motion_image) VALUE (f : set{frame})
  --The set of frames is defined as a full_motion_image

  CONCEPT image(fmi : full_motion_image) VALUE (f : frame)
  --given a full motion image this will return the current frame.
END
----------------------------------------------------------------------------------------------------
--The image system interface should get sets of frames from the image system for dipslay.  The sets of
--images should be displayed at a rate of 25 frames per second.
----------------------------------------------------------------------------------------------------
MACHINE image_system_interface

   STATE (video : full_motion_image)
   INVARIANT fmi_on
   INITIALLY  SOME(fmi : full_motion_image :: new_image(fmi))

   MESSAGE update_image REPLY (v : full_motion_image) WHERE v = video
      --Full motion video returned is current full motion image held by image system interface

  MESSAGE freeze_image REPLY(f : full_motion_image) where f = freeze_image(video, f)
    --f returned is a full motion image of the same frame

   MESSAGE new_image(fmi : full_motion_image) TRANSITION video = update_image(fmi)
      --new full motion image sent to interface is accepted as current full motion image held by interface

   CONCEPT fmi_on : boolean
     WHERE fmi_on <=>  ALL(fmi : full_motion_image :: frame_rate(fmi) >= 25 &
image_system_detected)
```

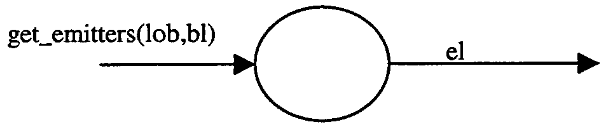
```
    --full motion image is present if the frame rate is greater than 25 and the image system is detected.

  CONCEPT image_system_detected : boolean
   --if image system is detected then this is true
END
```

APPENDIX D. STIMULUS-RESPONSE DIAGRAMS (SPEC)

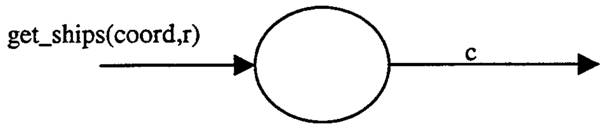
Image View



Emitter View



Ship Position View



Setup View

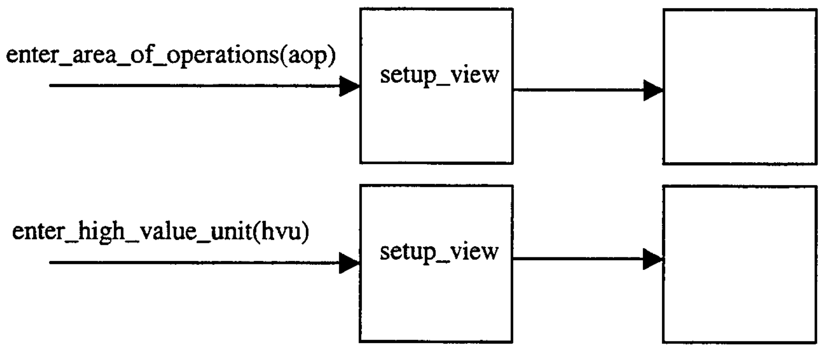
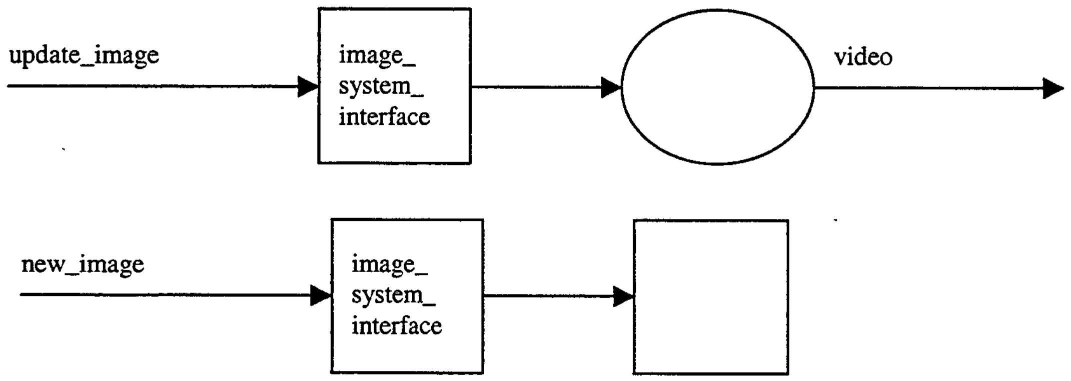
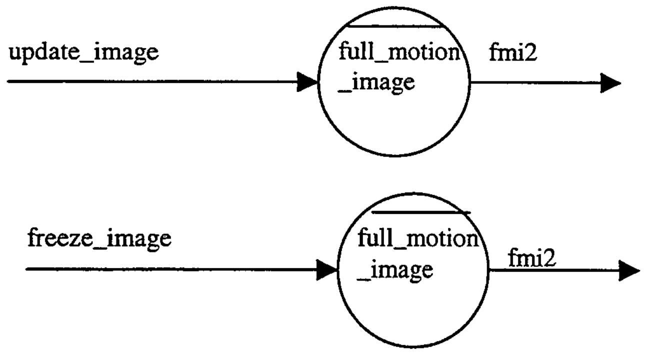


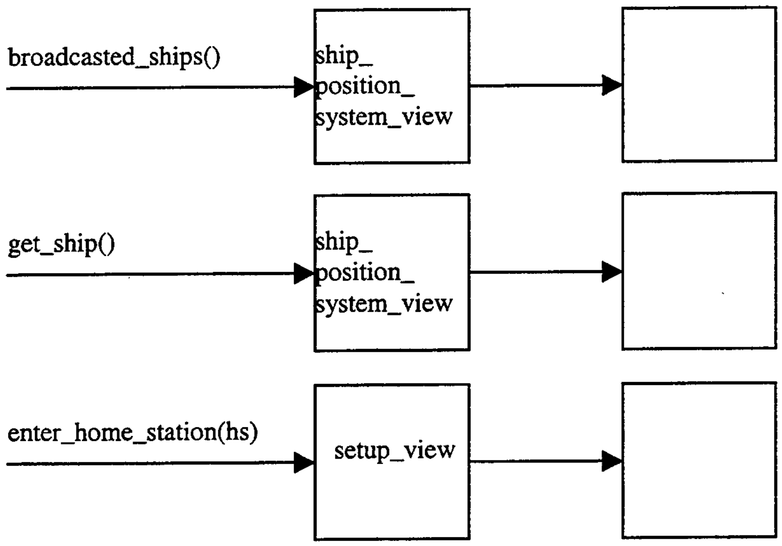
Image System Interface



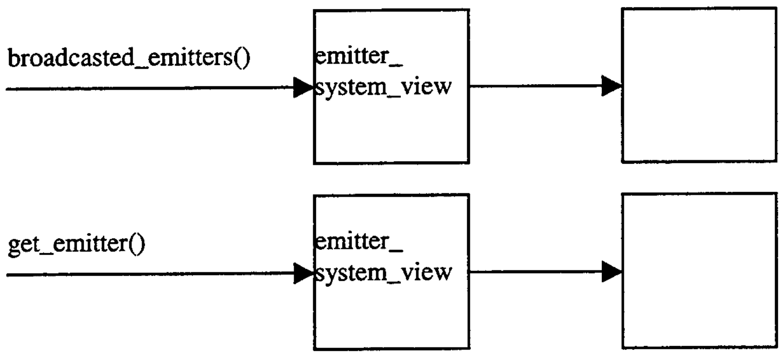
Full Motion Image



Ship Position System View

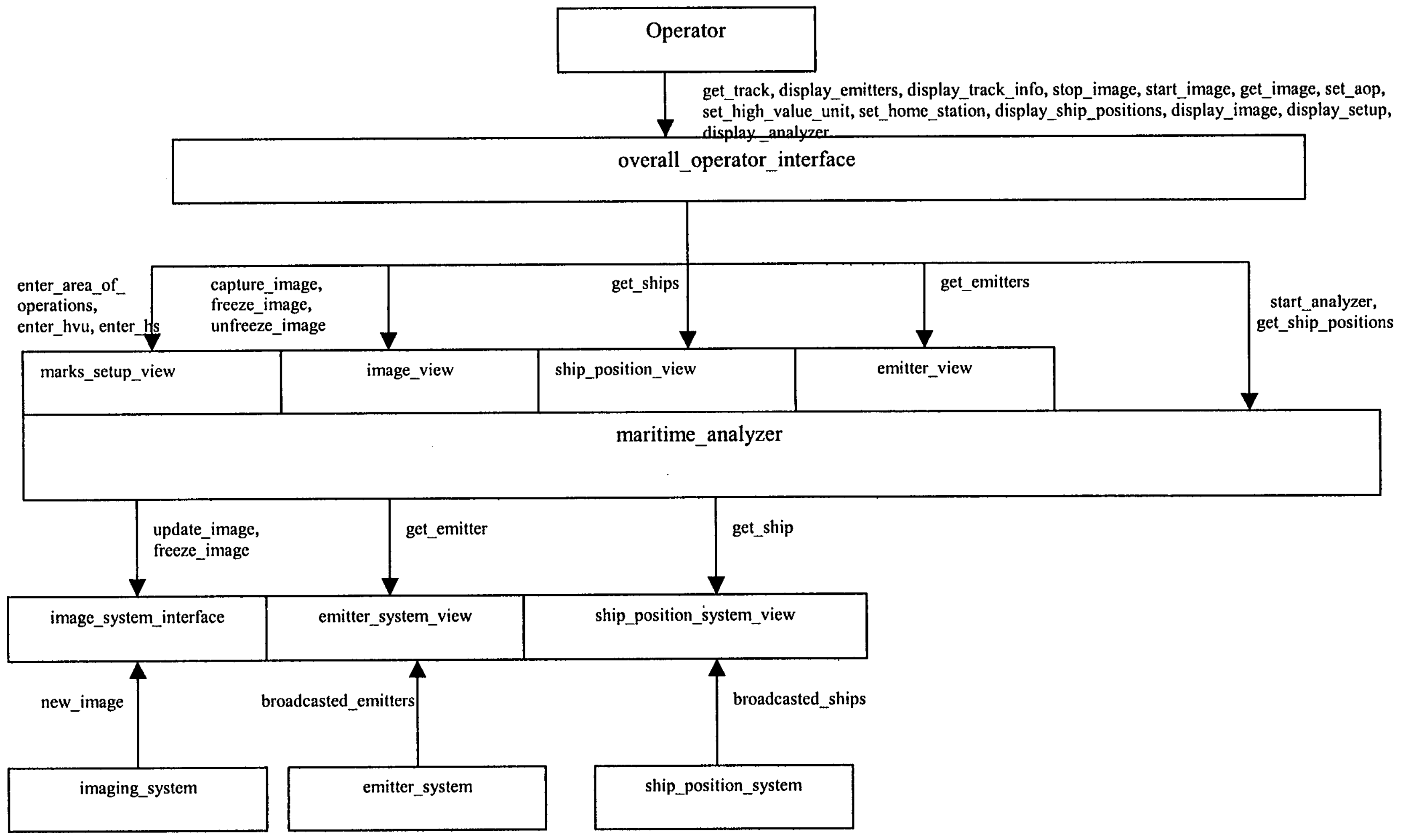


Emitter System View



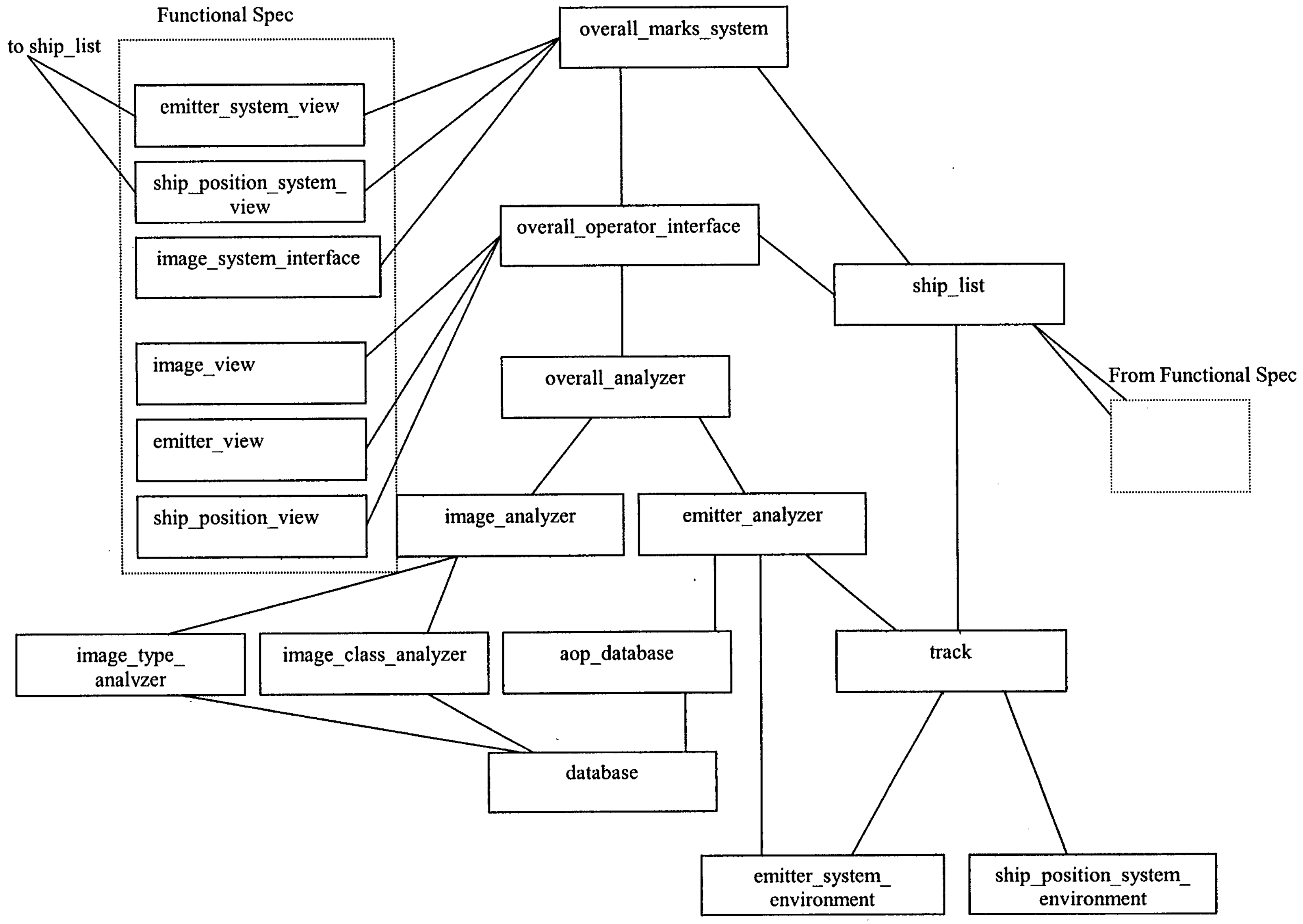
THIS PAGE IS INTENTIONALLY LEFT BLANK

APPENDIX E. MESSAGE FLOW DIAGRAM (SPEC)


Operator
get_track, display_emitters, display_track_info, stop_image, start_image, get_image, set_aop, set_high_value_unit, set_home_station, display_ship_positions, display_image, display_setup, display_analyzer
overall_operator_interface
enter_area_of_ operations, enter_hvu, enter_hs
capture_image, freeze_image, unfreeze_image
get_ships
get_emitters
start_analyzer, get_ship_positions
marks_setup_view
image_view
ship_position_view
emitter_view
maritime_analyzer
update_image, freeze_image
get_emitter
get_ship
image_system_interface
emitter_system_view
ship_position_system_view
new_image
broadcasted_emitters
broadcasted_ships
imaging_system
emitter_system
ship_position_system

THIS PAGE IS INTENTIONALLY LEFT BLANK

APPENDIX F. ARCHITECTURAL DESIGN MODULE DEPENDENCY



THIS PAGE IS INTENTIONALLY LEFT BLANK

APPENDIX G. ABSTRACT ARCHITECTURAL DESIGN (SPEC)

```
MACHINE overall_marks_system

  INHERIT overall_operator_interface
  INHERIT ship_list
  INHERIT emitter_system_view
  INHERIT image_system_interface
  INHERIT ship_position_system_view

  MESSAGE start_marks
    SEND update_list TO ship_list
    SEND update_ship_emitters TO ship_list
  --update_list and update_ship_emitters are called to initiate updating those systems.  With the inherit
  --overall_operator_interface, image_system_interface emitter_system_view, and
  --ship_position_system_view are initialized (i.e. turned on)

END

----------------------------------------------------------------------------------------------------
-- Th Emitter Analyzer will analyze a given list of emitters against a set of ships held by the Area of
-- Operations Database.  The function "analyze_emitters" will take all the ships listed in the AOP Database
-- and match the given emitter list against the ship's emitters in the AOP Database.  If a match is found a
-- confidence meter reading is generated on how much the lists match. The set returned contains ships
-- matched with confidence meters.
----------------------------------------------------------------------------------------------------
FUNCTION emitter_analyzer{aop : area_of_operation}

  INHERIT track
  INHERIT aop_database{aop}
  INHERIT emitter_system_environment
  INHERIT confidence_meter_definitions

  MESSAGE analyze_emitters(el : emitter_list) REPLY (s : set{ship_emitter_info})
    WHERE ALL(dps : set{database_pair} :: dps = get_aop(aop) => ALL(x : database_pair :: x IN dps =>
          SOME(sei : ship_emitter_info :: emitter_list_match(el,x) =>
                 sei.s = x.s, sei.cm = ship_confidence(el, x) &  sei IN s)))
  --s1 will contain the set of all ships have emitters that match the given emitter list and a confidence meter
  --associated with how well the lists match.

  CONCEPT ship_emitter_info : type
    WHERE ship_emitter_info = tuple {s :: ship, cm :: confidence_meter)

  CONCEPT emitter_list_match(el : emitter_list, dp : database_pair) VALUE (b : `  oolean)
  --b if emitters in the emitter list match the emitters a ship would would emit

  CONCEPT ship_confidence(el : emitter_list, dp : database_pair) VALUE (cm : confidence_meter)
  --cm is between 0.0 and 1.0 depending upon match of emitters in list to possible emitters on ship
END


----------------------------------------------------------------------------------------------------
-- The image analyzer will take in a captured image and check the image for ship type and then ship class.
-- It is finally analyzed for identification with a set of ship,confidence meter pairs returned.
```

```
----------------------------------------------------------------------------------------------------------------
FUNCTION image_analyzer{aop : area_of_operation}

  INHERIT image_type_analyzer{aop}
  INHERIT image_class_analyzer{aop}
  INHERIT analyzer_definitions

  MESSAGE analyze_image{image : frame} REPLY (s : set{ship_id_tuple})
    WHERE ALL(itt : set{image_type_tuple}, ict : set{ship_id_tuple}
                    :: itt = analyze_image_type(image) =>  ict = analyze_image_class(image, itt)
                    => SOME(imd : image_id :: imd = image_ship_match(ict)=> imd IN s))

 CONCEPT image_ship_match(ict : set{ship_id_tuple}) VALUE (s : set{ship_id_tuple})
 --Correlates the set of ship class/type with confidence meters to a set of ship identifications with a
 --confidence meter
END

----------------------------------------------------------------------------------------------------------------
-- The image type analyzer takes a frame and transforms it to an image type that can be analyzed against
-- the database of image types.  The returned set contains a set of tuples of  imaget_type_tuple.  This
-- function is used to narrow the candidates for identification down from the set of all ships operating in the
– AOP to the set of specific ship types operating in the AOP.
----------------------------------------------------------------------------------------------------------------
FUNCTION image_type_analyzer{aop : area_of_operation}

  INHERIT aop_database{aop}
  INHERIT emitter_system_environment
  INHERIT confidence_meter_definitions
  INHERIT analyzer_definitions

  MESSAGE analyze_image_type(image : frame)REPLY (s : set{image_type_tuple})
            --list of ships, type, class, image, confidence meter
    WHERE ALL(it : image_type :: it = transform_image_to_type(image) =>
          ALL(dps : set{database_pair} :: dps = get_aop(aop) => ALL(x : database_pair :: x IN dps =>
            SOME(its : image_type_tuple, cm : confidence_meter :: image_type_match(it,x.it) >= 0.4 =>
              its.sh = x.s, its.st = x.st, its.sc = x.sc, its.ic = x.ic, its.cm = image_type_match(image,x.it) &
              its IN s))))
 --Matches the transformed frame against the saved image types in the database.

  CONCEPT image_type_match(image1 image2 : image_type) VALUE  (cm : confidence_meter)

  CONCEPT transform_image_to_type(image : frame) VALUE (type_image : image_type)
  --This transform will convert a given frame to an image_type

END

----------------------------------------------------------------------------------------------------------------
-- The image class analyzer takes a frame and a set of image_type_tuple.  It transforms the frame to a class
-- type that can be analyzed against the set of class types contained in the given set.  The returned set
-- contains a set of tuples of image_class_tuple This function is used to narrow the candidates for
-- identification down from the set of ship types operating in the AOP to the set of specific ship classes by
--  type operating in the AOP
----------------------------------------------------------------------------------------------------------------
FUNCTION image_class_analyzer
```

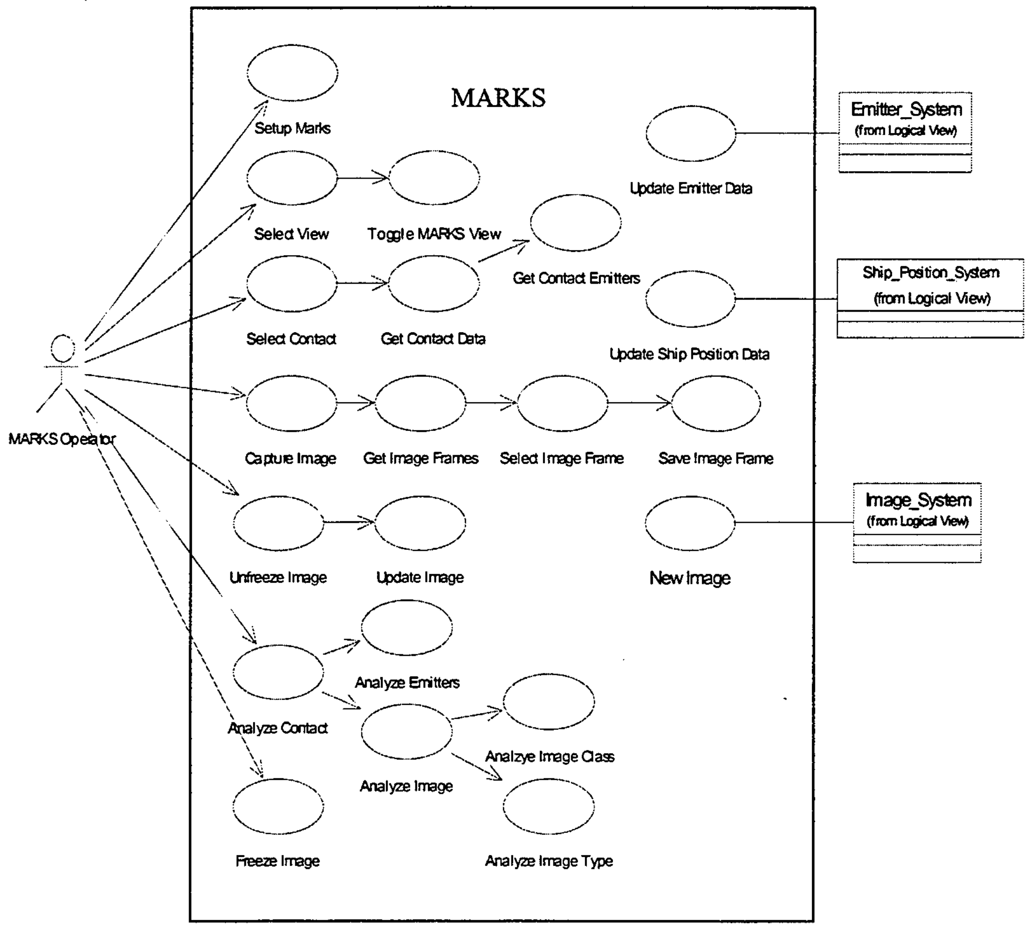
```
 INHERIT analyzer_definitions

 MESSAGE analyze_image_class(image : frame, itt : set{image_type_tuple})
   REPLY (s : set{image_type_list}
     WHERE ALL(ic : image_class :: ic = transform_image_to_class(image) =>
             ALL(x : image_type_tuple :: x IN  itt =>  SOME(ict : ship_id_tuple, cm : confidence_meter ::
               image_type_match(ic,x.ic) >= 0.4 =>  ict.sh = x.s, ict.st = x.st, ict.sc = x.sc,
               ict.cm = image_class_match(ic,x.ic) & ict IN s))))

 MESSAGE `  oolean_image_class(image : frame, itl : list{image_type_list})
   REPLY e –list of ship,type,class,confidence_meter


 CONCEPT image_class_match(image1 image2 : image_class) VALUE  (cm : confidence_meter)

 CONCEPT transform_image_to_class(image :frame) VALUE  (class_image : image_class)

END

----------------------------------------------------------------------------------------------------------------
--Starts the analysis using the image and the track data
----------------------------------------------------------------------------------------------------------------
MACHINE overall_analyzer

 INHERIT analyzer_definitions
 INHERIT confidence_meter_definition
 INHERIT emitter_analyzer
 INHERIT image_analyzer

 STATE (s im em : set{ship_id_tuple})
 INVARIANT true
 INITIALLY s = []

 MESSAGE start_analyzer(image : frame, t : track)REPLY s
  TRANSITION im = analyze_image(image), em = analyzer_emitters(t.corr_emitters),
          s = correlate_em_im(im,em)
 CONCEPT correlate_em_im(im em: set{ship_id_tuple})  VALUE (s1 : set{ship_id_tuple})
 --Correlates the entries in the im set and em set and returns a new set of ship_id_tuple

END

----------------------------------------------------------------------------------------------------------------
--Analyzer definitions
----------------------------------------------------------------------------------------------------------------

DEFINITION analyzer_definitions


  CONCEPT image_type_tuple : type
    WHERE image_type_tuple = tuple{sh :: ship, st :: ship_type, sc :: ship_class, ic :: image_class,
                        cm :: confidence_meter}

  CONCEPT ship_id_tuple : type
   WHERE ship_id_tuple = tuple {sh :: ship, st :: ship_type, sc :: ship_class, cm :: confidence_meter}
```

```
END

--------------------------------------------------------------------------------------------------------
-- track is a TYPE that holds all the information needed on
-- a contact held in the ship position system
--------------------------------------------------------------------------------------------------------
TYPE track

   MODEL(tn    : track_number, id    : id, tp    : type, cl    : class, cat   : category,
                  si    : ship_position_system_id, tg    : tag, coord : coordinate, c : course, s : speed,
                  corr_emitters : emitter_list)
    INVARIANT true

   MESSAGE get_symbol_info REPLY (s : symbol_info)
       WHERE  s.tn = t.tn, s.si = t.si, s.cat = t.cat,  s.coord = t.coord
          -- returns all the info needed to display the track's symbol

   MESSAGE update(spt : ship_position_type) REPLY (ut : track)
      WHERE ut.tn = spt.tn, ut.cat = spt.cat, ut.id = spt.i, ut.tg = spt.t, ut.coord = spt.coord
          -- updates a track or creates a new track based on new ship_position_type info


  MESSAGE update(e : emitter_data_type)
          -- updates the list of emitters that correlate to the track if the emitter doesn't correlate to the ship's
          --position, then remove the emitter from corr_emitters
    WHEN (e.el IN corr_emitters) & (~correlates(e)) REPLY (ut : track)
      WHERE t.tn = tn, ut.id = id, ut.tp = tp, ut.cl = cl,  ut.cat = cat, ut.si = si, ut.tg = tg,
                  ut.coord = coord, ut.c = c, ut.s = s,  ut.corr_emitters = remove(e.estn, corr_emitters)
             WHEN  ~(e.el IN corr_emitters) & (correlates(e))
               -- if the emitter does ` oolean` e to the ship's position, then add it to the list of emitters that
               --coorelate to that ship's position
              REPLY (ut : track)
                  WHERE t.tn = tn, ut.id = id, ut.tp = tp, ut.cl = cl,  ut.cat = cat, ut.si = si, ut.tg = tg,
                         ut.coord = coord, ut.c = c, ut.s = s,
                         ut.corr_emitters = bind(e.estn, e.emitter,  corr_emitters)

             OTHERWISE REPLY (ut : track)
                 WHERE  ut.tn = tn, ut.id = id, ut.tp = tp, ut.cl = cl, ut.cat = cat, ut.si = si, ut.tg = tg,
                     ut.coord = coord, ut.c = c, ut.s = s, ut.corr_emitters = corr_emitters

  CONCEPT symbol_info : type
                  -- symbolic info is all the track information the gui needs to display the track's symbol
    WHERE symbol_info = tuple{tn :: track_number,  si :: ship_position_system_id, cat :: category,
                                     coord :: coordinate }

  CONCEPT emitter_list : map{emitter_system_track_number, emitter}

  CONCEPT correlates(e: emitter_data_type) VALUE (b : ` oolean)
     WHERE b ⇔ abs(bearing(e.el.bl, coord) – e.el.lob) <= bearing_accuracy

  CONCEPT bearing(from to : coordinate) VALUE (b : line_of_bearing)
    -- bearing converts two coordinates into a line_of_bearing from the first coordinate to the second
    --coordinate

END
```

```
---------------------------------------------------------------------------------------------------------------
-- The ship list holds the ships received from the ship position system interface.
---------------------------------------------------------------------------------------------------------------
MACHINE ship_list

  INHERIT map
  INHERIT ship_position_system_environment
  INHERIT track

  STATE(m: map{track_number, track})
  INVARIANT true
  INITIALLY m = []

  MESSAGE get_symb_info   -- returns a map{track_number, symbol_info} of all the tracks in m
    REPLY(symb_map : map{track_number, symbol_info} )
     WHERE ALL(t : track_number :: t IN m => bind(t, get_symbol_info(t), symb_map))

   MESSAGE get_track_info(t_num : track_number) – returns track info on specified track
     WHEN t_num IN m    REPLY(t : track) WHERE t = m[t_num]
     OTHERWISE REPLY EXCEPTION no_such_track

   MESSAGE update_list(spt : ship_position_type) – updates a particular track
     WHEN spt.tn IN m                              -- if the track already exists, update it
          TRANSITION m = bind(spt.tn, update(spt),*m[spt.tn])
      OTHERWISE  -- if the track doesn't exist yet, convert the ship_position_type
                       -- to a track and then add it to the list
          TRANSITION m = bind( spt.tn, update(spt), *m)

   MESSAGE update_ship_emitters(emitter : emitter_data_type, position : coordinate)
     REPLY (m : map{track_number, track}) WHERE ALL(t: track_number :: t IN m)
END
```

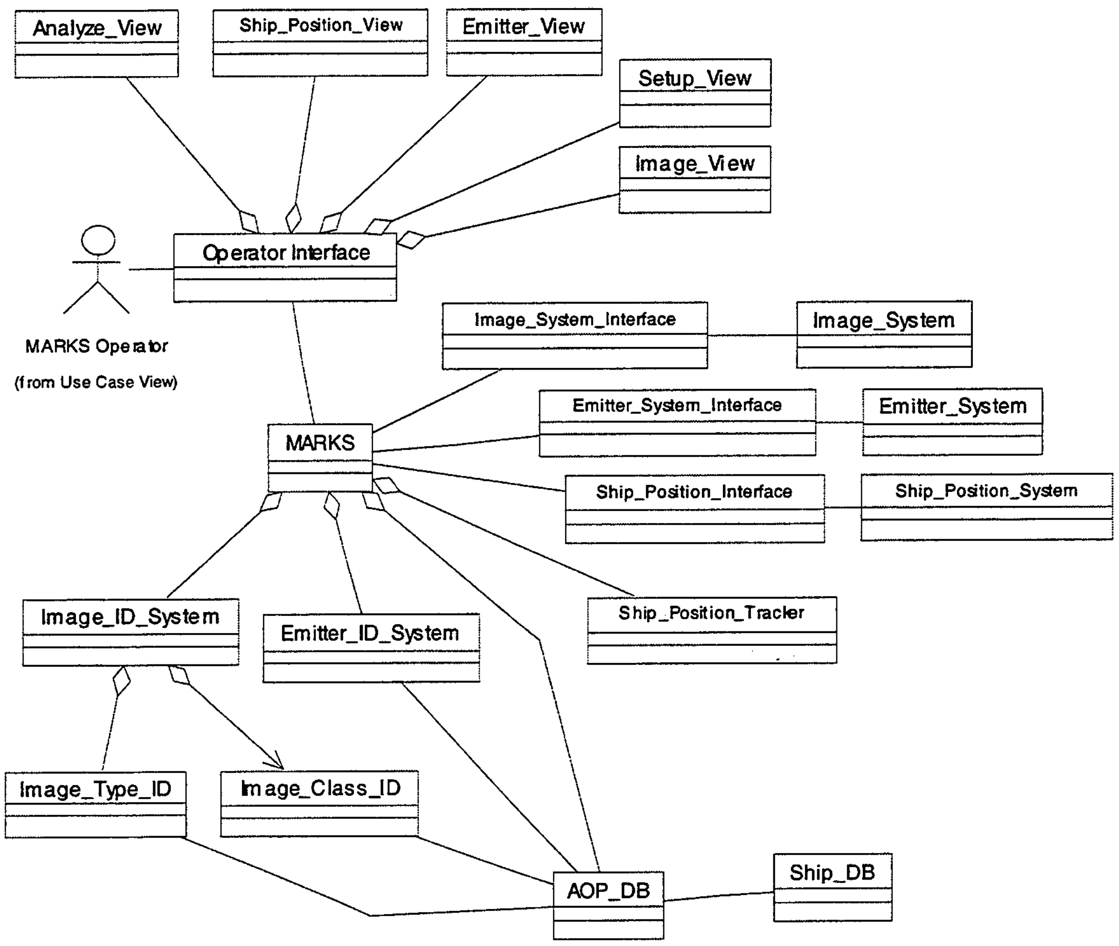
THIS PAGE IS INTENTIONALLY LEFT BLANK

APPENDIX H. USE-CASE DIAGRAM (UML)



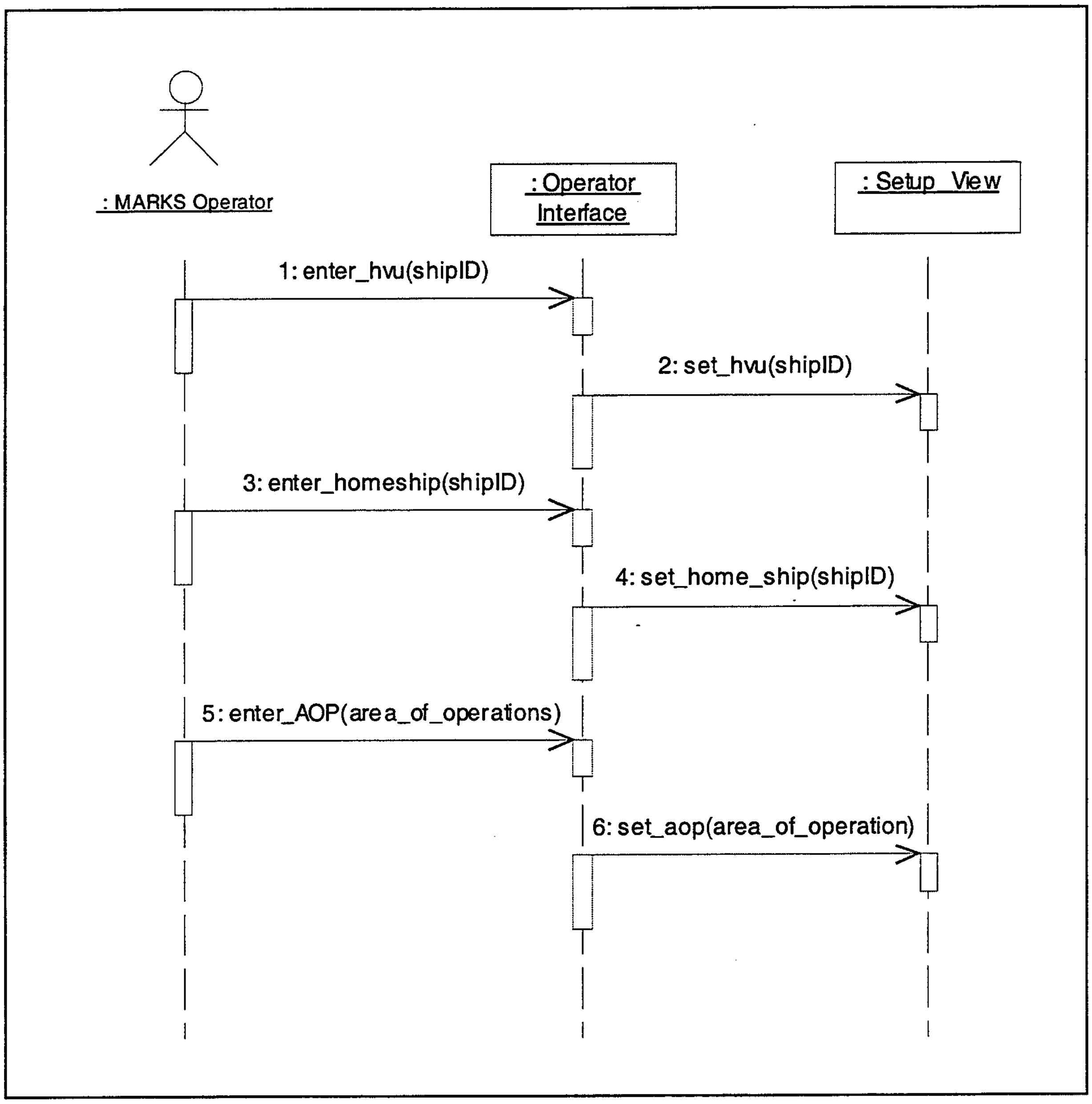
THIS PAGE IS INTENTIONALLY LEFT BLANK

APPENDIX I. CONCEPTUAL MODEL (UML)

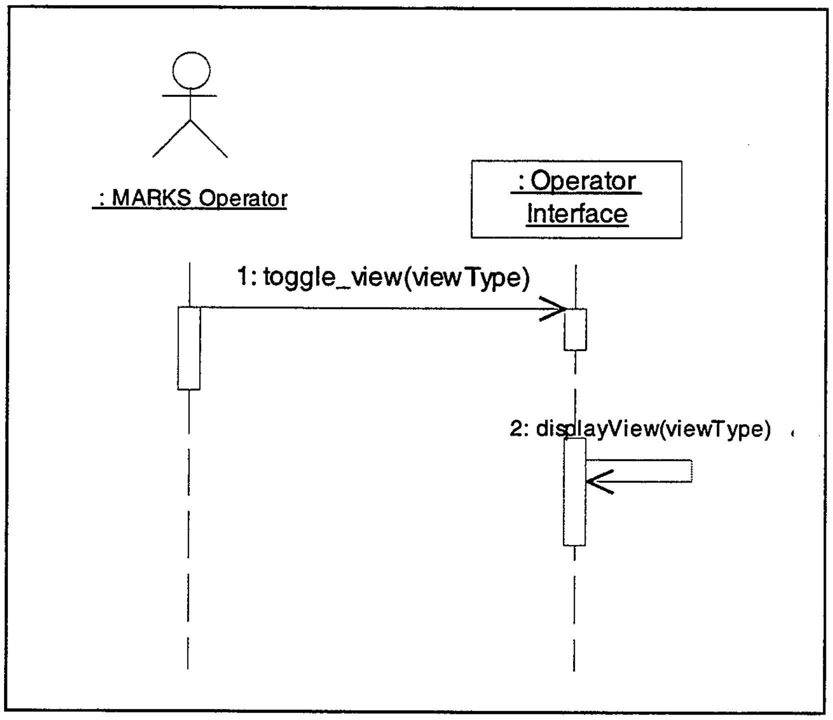


THIS PAGE IS INTENTIONALLY LEFT BLANK

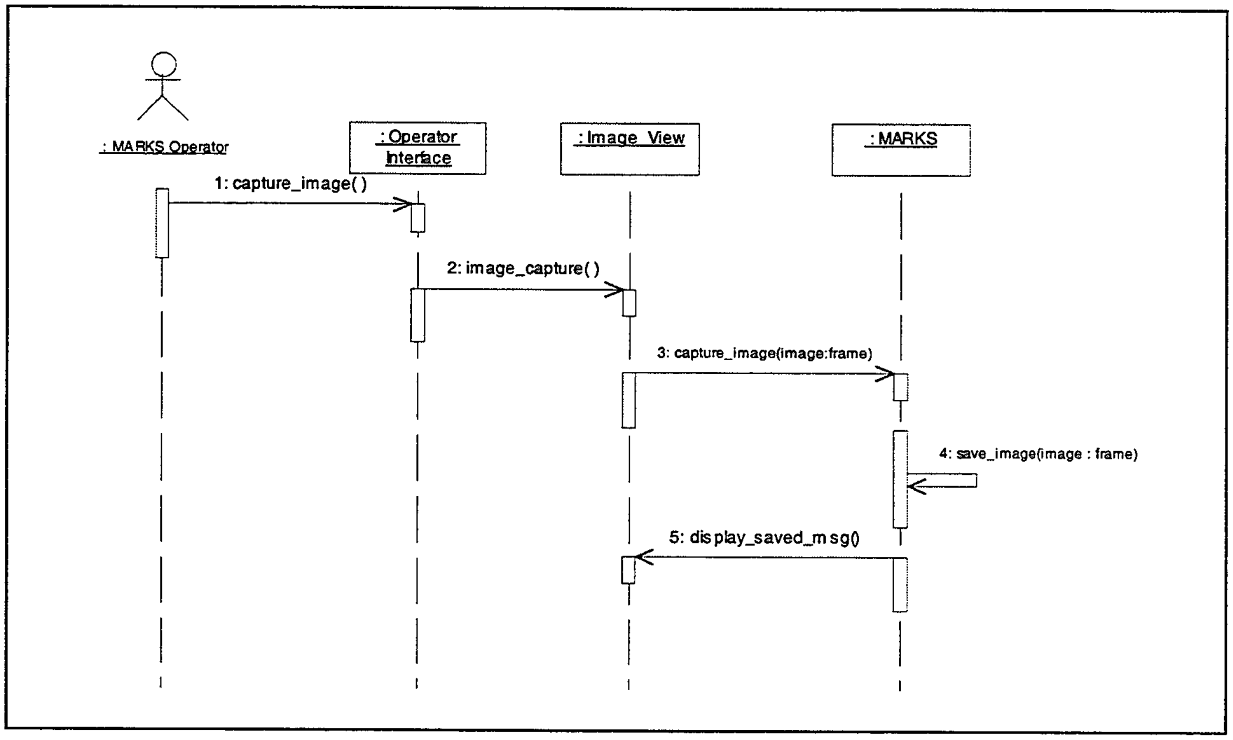
APPENDIX J. SYSTEM-SEQUENCE DIAGRAMS (UML)



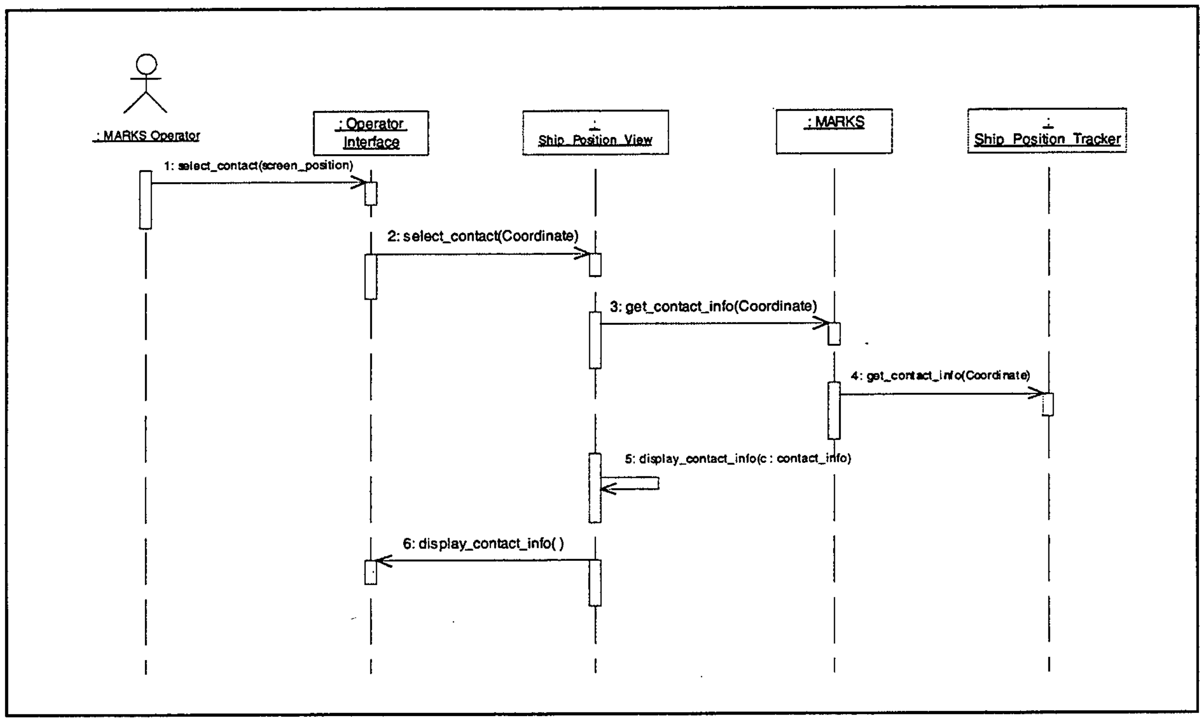
Setup MARKS



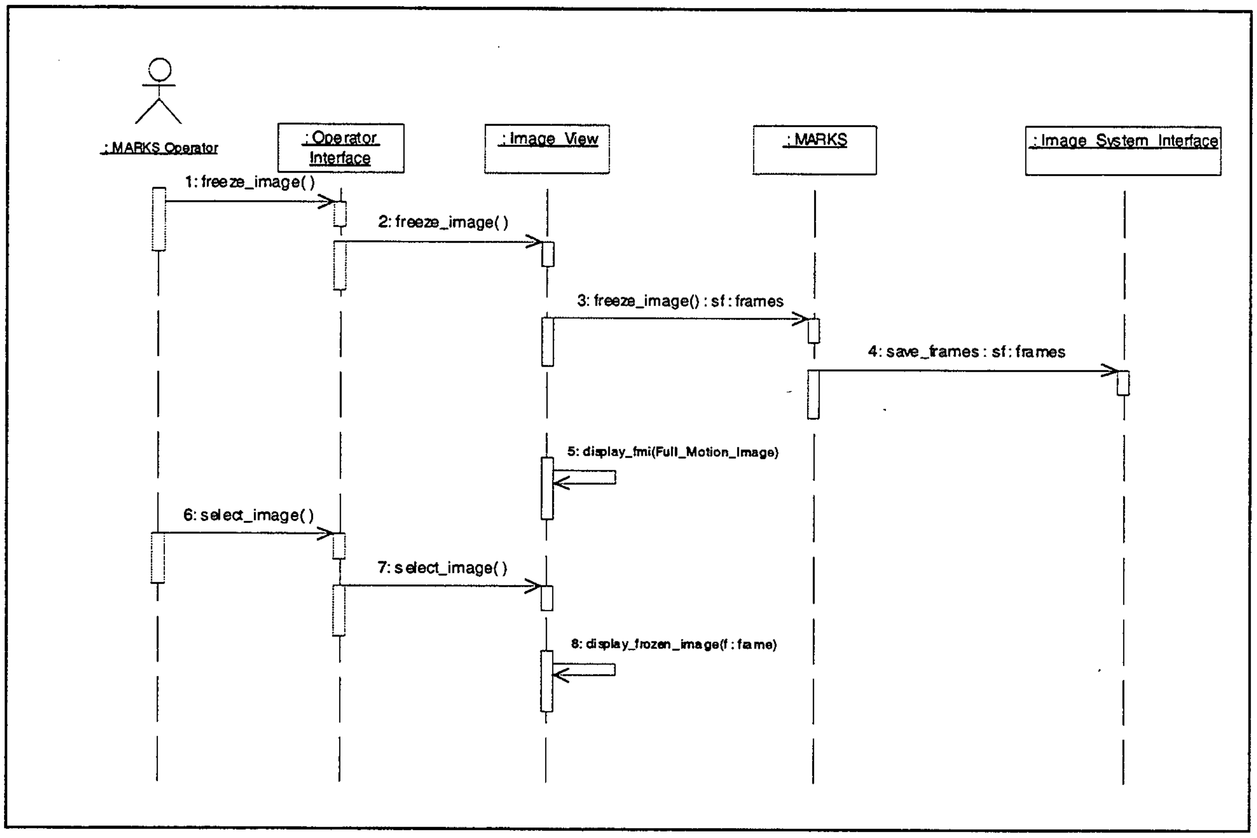
Select View



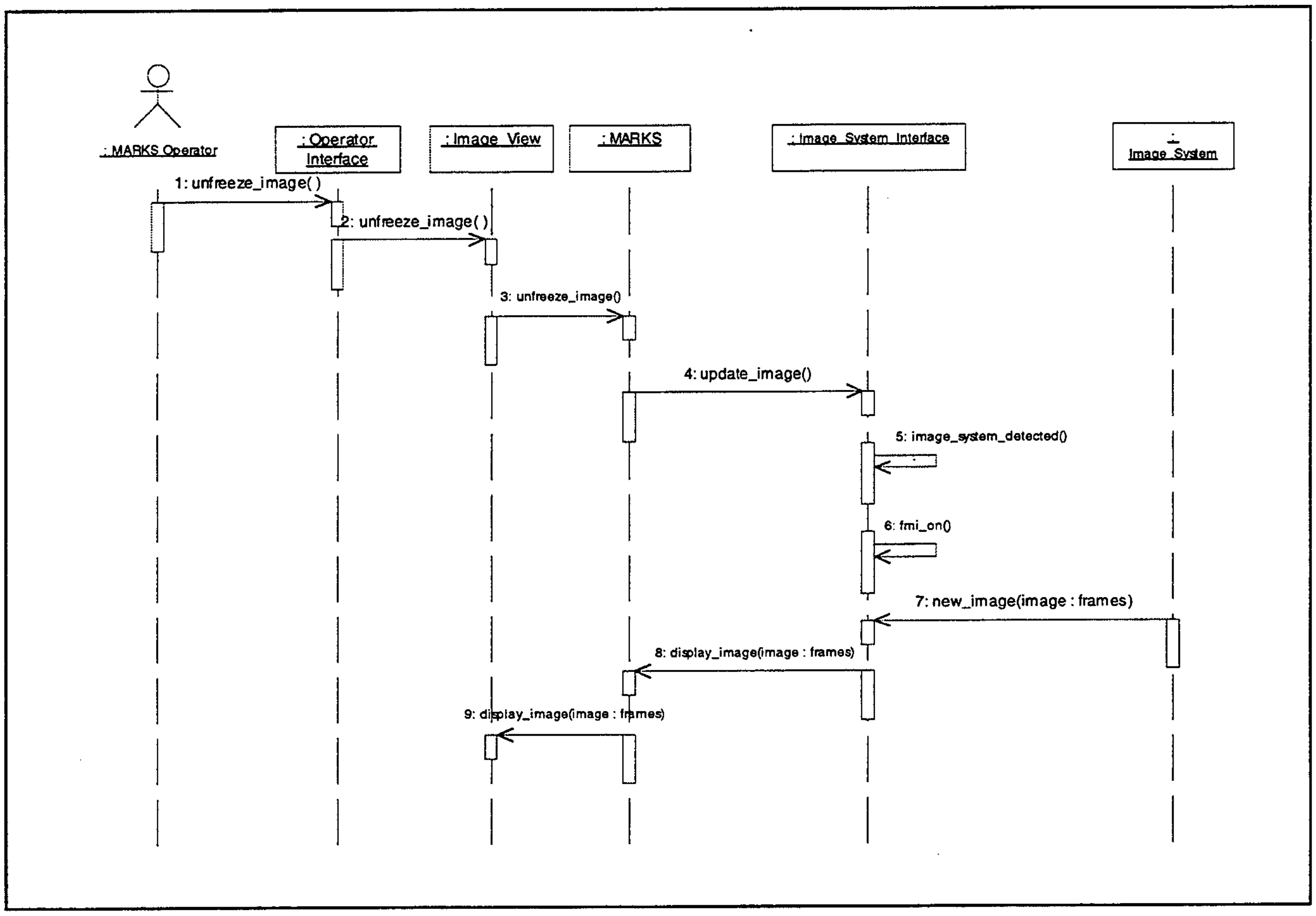
Capture Image


: MARKS Operator
: Operator Interface
: Ship Position View
: MARKS
: Ship Position Tracker
1: select_contact(screen_position)
2: select_contact(Coordinate)
3: get_contact_info(Coordinate)
4: get_contact_info(Coordinate)
5: display_contact_info(c : contact_info)
6: display_contact_info( )

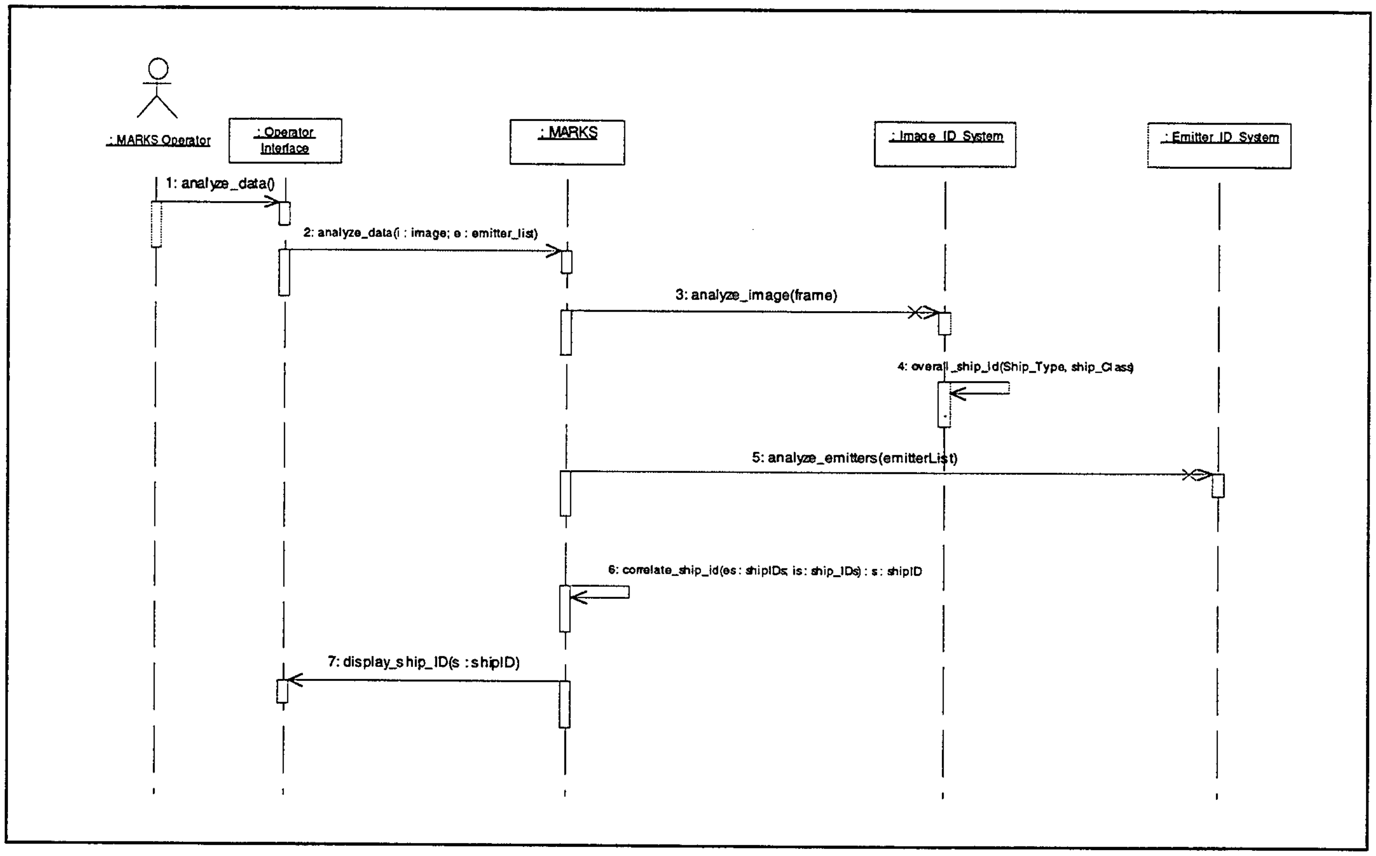
Select Contact



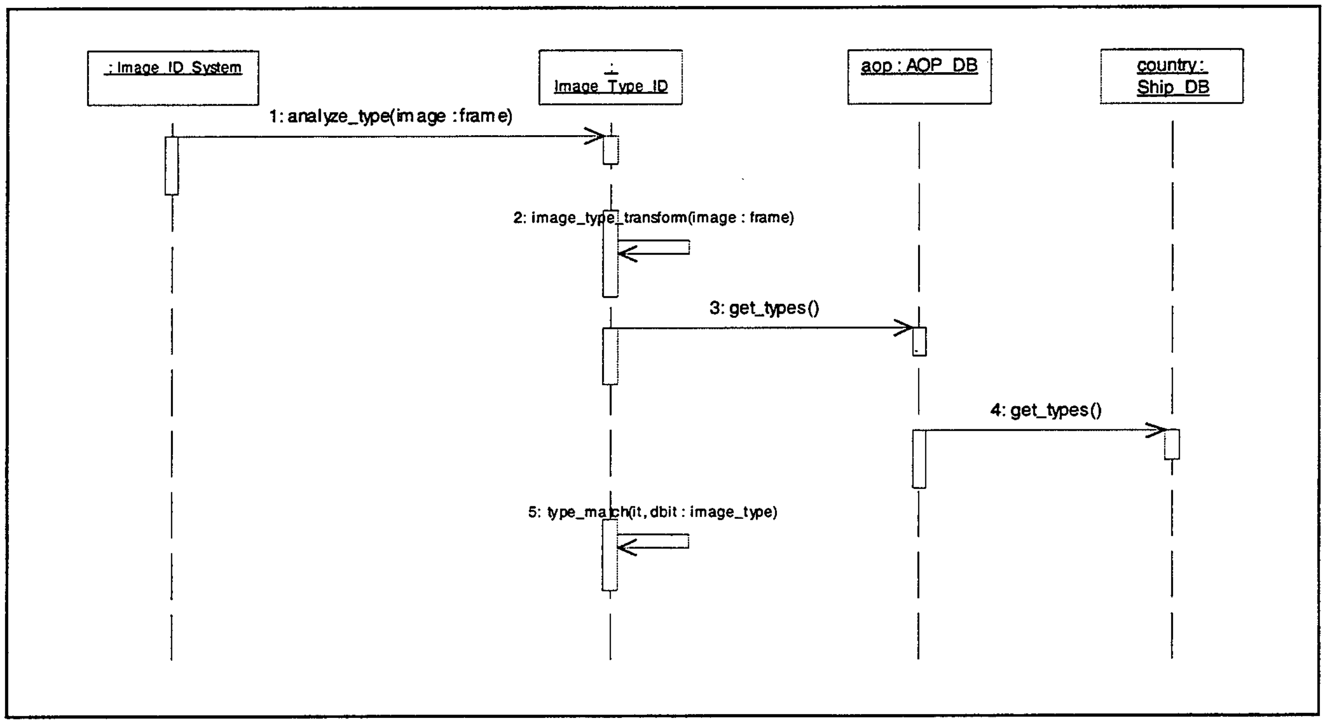
Freeze Image



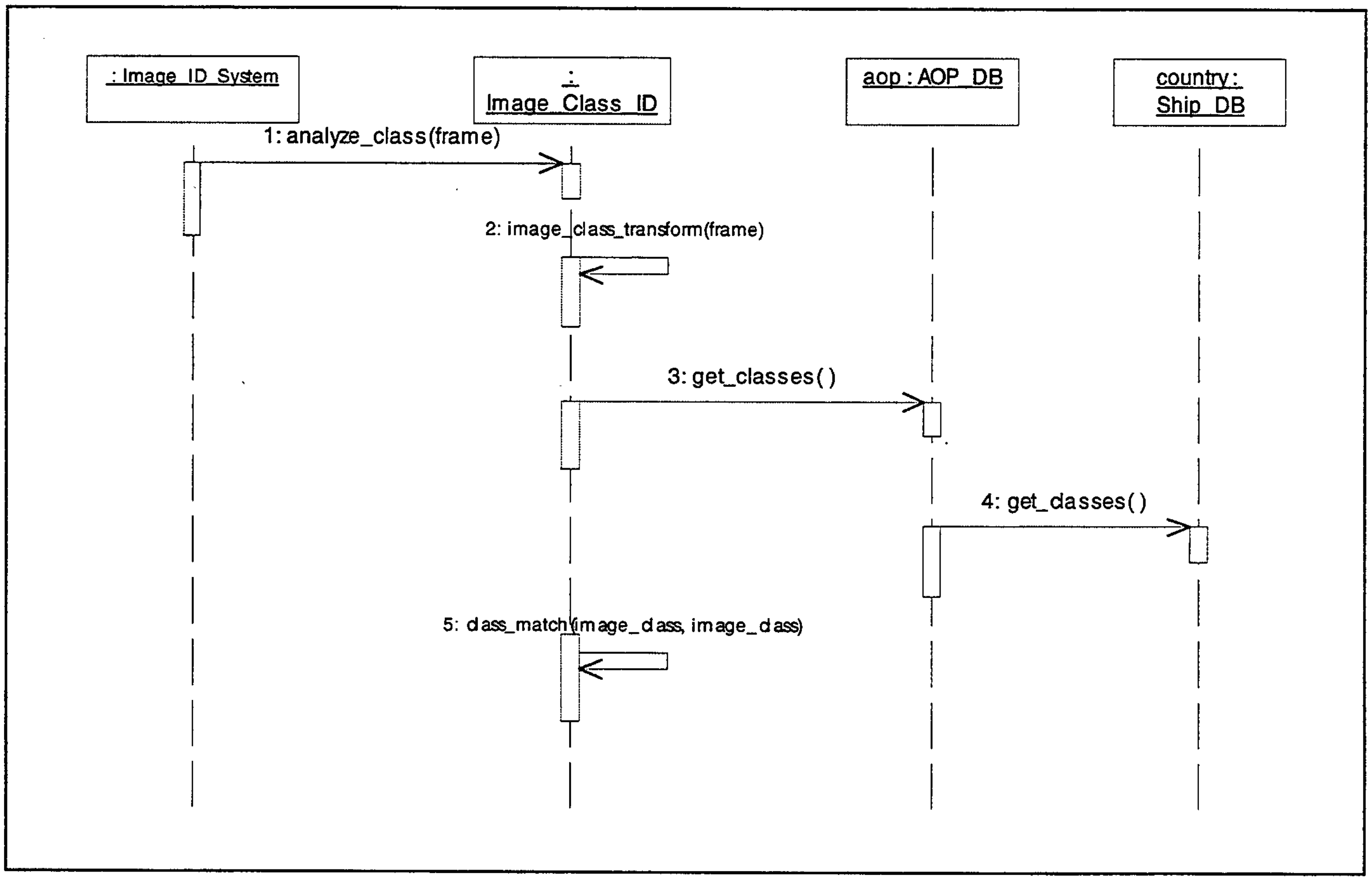
Unfreeze Image


: MARKS Operator
: Operator Interface
: MARKS
: Image ID System
: Emitter ID System
1: analyze_data()
2: analyze_data(i : image; e : emitter_list)
3: analyze_image(frame)
4: overall_ship_id(Ship_Type, ship_Class)
5: analyze_emitters(emitterList)
6: correlate_ship_id(es : shipIDs; is : ship_IDs) : s : shipID
7: display_ship_ID(s : shipID)

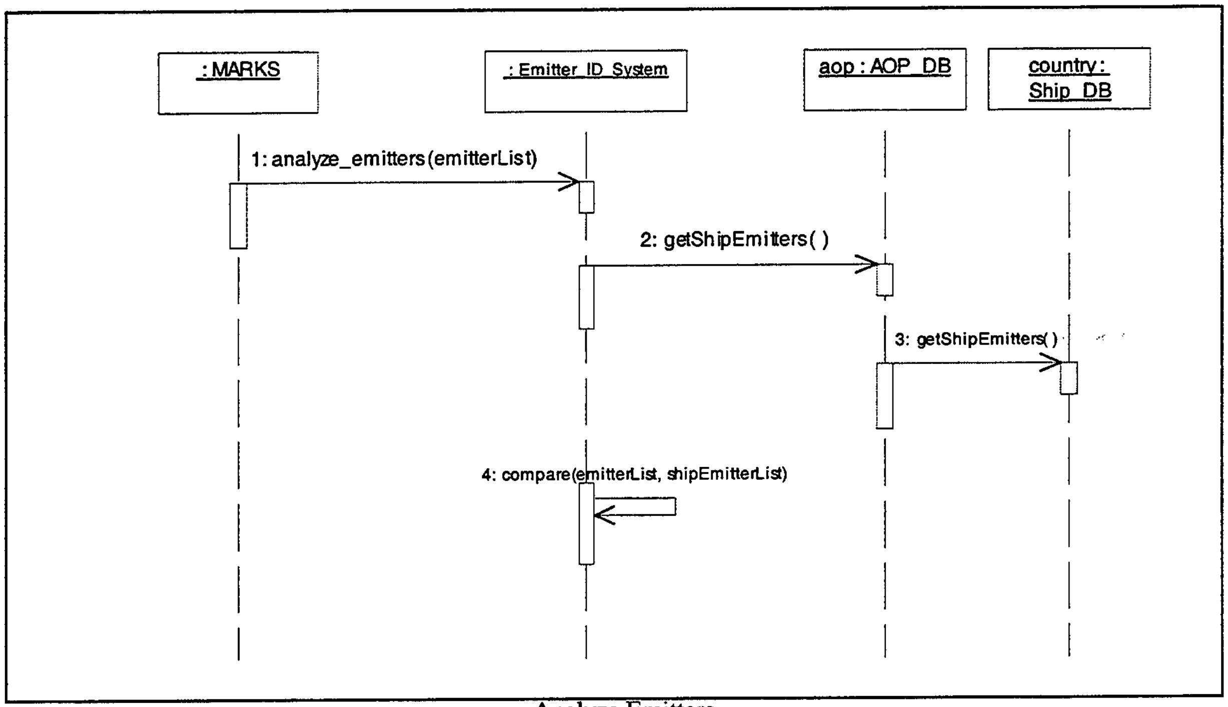
Analyze Data


: Image ID System
:
Image Type ID
aop : AOP DB
country :
Ship DB
1: analyze_type(image : frame)
2: image_type_transform(image : frame)
3: get_types()
4: get_types()
5: type_match(it, dbit : image_type)

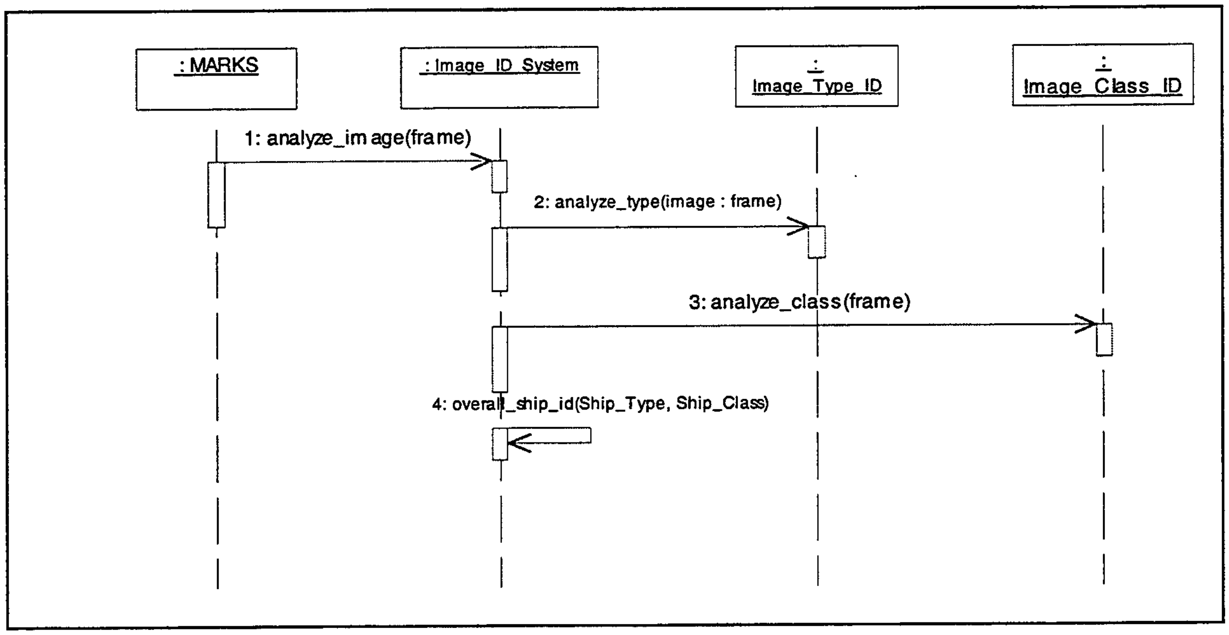
Analyze Type



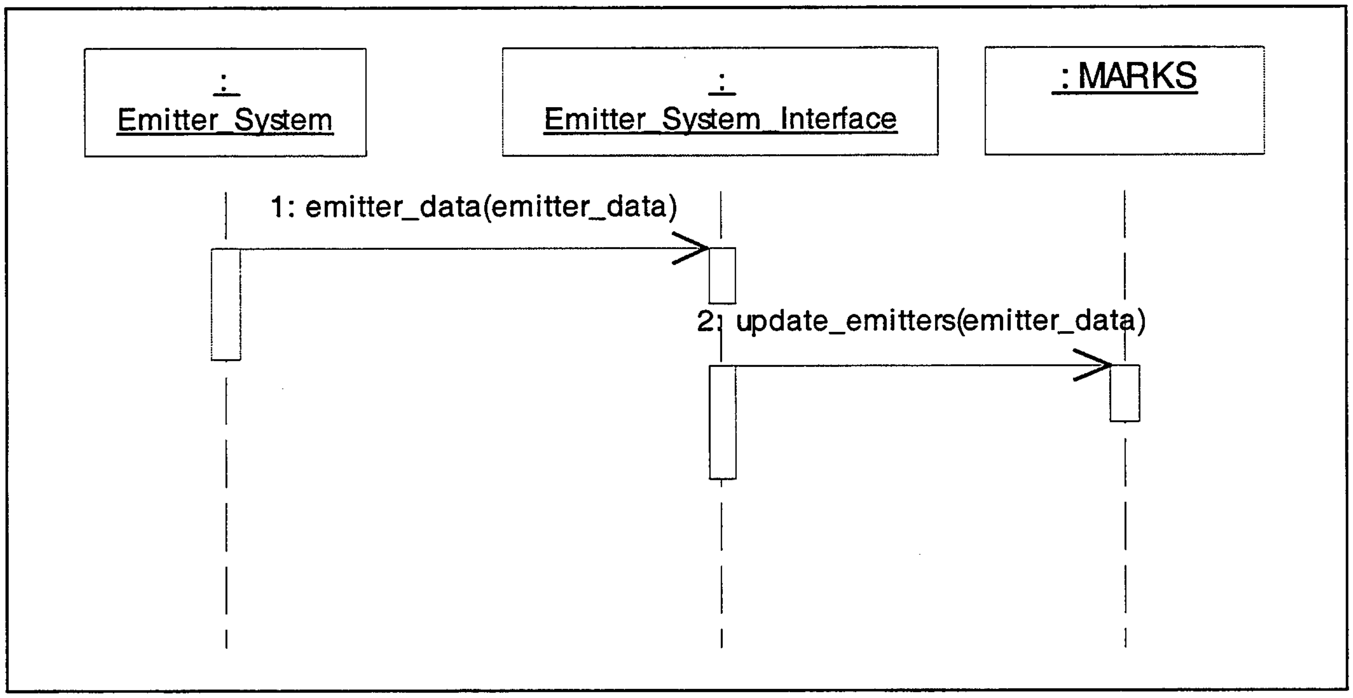
Analyze Class



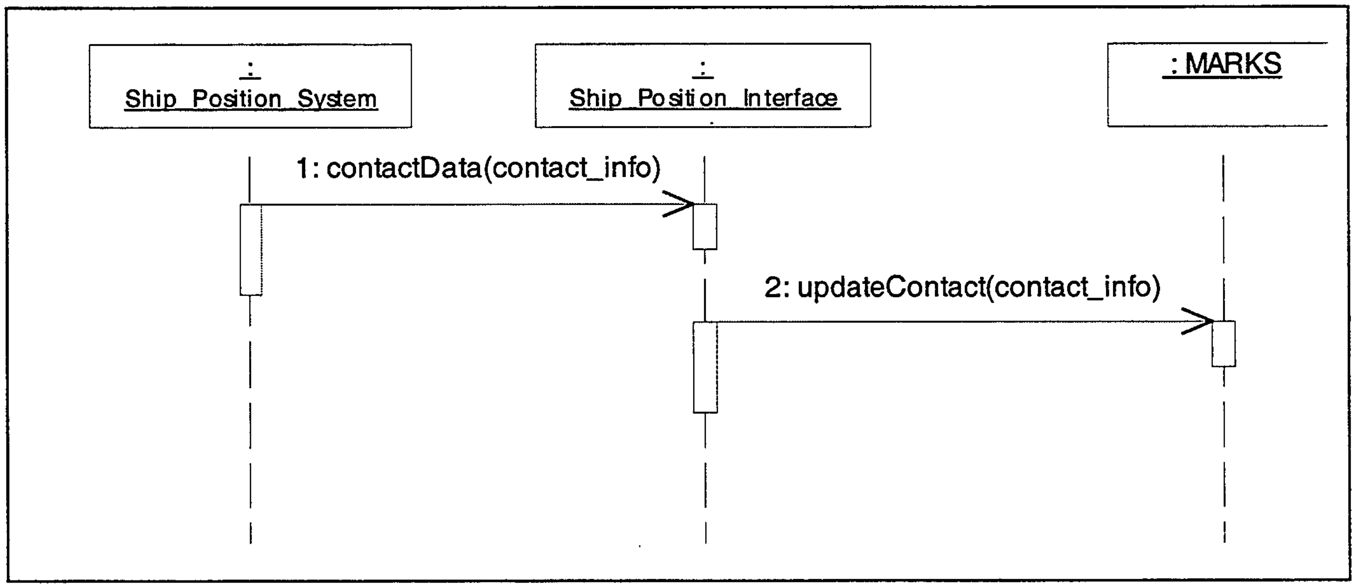
Analyze Emitters



Analyze Image


: Emitter_System
: Emitter_System_Interface
: MARKS
1: emitter_data(emitter_data)
2: update_emitters(emitter_data)

Update Emitter Data


: Ship_Position_System
: Ship_Position_Interface
: MARKS
1: contactData(contact_info)
2: updateContact(contact_info)

Update Ship Position Data

APPENDIX K. GLOSSARY, ACRONYMS, AND ABBREVIATIONS

AN/AAS-44V	2nd Generation FLIR deployed on SH-60B RDK and SH-60R helicopters
AN/ALQ-142	SH-60 based ESM System
AN/SLQ-32	Ship based ESM system
CIC	Command Information Center
COI	see Contact of Interest
Contact of Interest	Air, surface, or subsurface contact via radar, visual, or sonar that has not been identified
DOD	Department of Defense
Electronic Support Measures	A system that receives emitted radiation from other sources and interprets them
ESM	see Electronic Support Measures
ESMO	Ship based ESM operator
FLIR	Forward Looking Infrared
Graphical User Interface	Method of displaying and retrieving information from a user
GUI	see Graphical User Interface
Hawklink	Secure datalink used to exchange communications and data between LAMPS MKIII and its host ship
JMCIS COE	see Joint Maritime Command Information System Common Operating Environment
Joint Maritime Command Information System Common Operating Environment	Provides battle space management with a display of ship locations and identifications
Joint Operational Tactical System	Provides battle space management with a display of ship locations and identifications
JOTS	see Joint Operational Tactical System
LAMPS MKIII	see Light Airborne Multipurpose System Mark III
Light Airborne Multipurpose System Mark III	Helicopter that deploys from LAMPS equipped frigates, destroyers, and cruisers. Designated as SH-60B Seahawk. Other designations, SH-60B RDK and SH-60R
MARKS	Maritime Analyzer Recognition Knowledge System: This is the target recognition system being developed
NCTR	see Non-cooperative target recognition system
Non-cooperative target recognition system	Identification system that does not rely on an interrogation/response process
SENSO	SH-60B Sensor Operator
SO	SH-60B Sensor Operator
SPEC	A formal specification language developed by Dr. Luqi and Dr. Berzins of the Naval Postgraduate School
Surface Warfare Officer	An officer in the U.S. Navy that is an expert in ship board operations
SWO	see Surface Warfare Officer
Target of Interest	Air, surface, or subsurface contact that has been identified has hostile
TOI	see Target of Interest
Track	Contact held in MARKS
UML	Unified Modeling Language

VID	see visual identification
Visual Identification	The method of using a human being to identify a COI/TOI via visual means.
Z	Pronounced "Zed", a formal specification language

LIST OF REFERENCES

[1] N. Polmar, *The Naval Institute Guide to the Ships and Aircraft of the U.S. Fleet*, Fifth Edition, Naval Institute Press, March 01, 1995

[2] H.M. Holt, "Assessment of Fault Tolerant Computing Systems at NASA's Langley Research Center," IEEE Proceedings Aerospace Conference, February 8, 1997, pp. 541-49

[3] D. Hamilton, R. Covington, J. Kelley, "Experiences in Applying Formal Methods for the Analysis of Software System Requirements," Workshop on Industrial- Strength Formal Specifications Techniques, April 5, 1995, pp. 30-43

[4] S. Easterbrook, R. Lutz, R. Convington, J Kelley, Y. Ampo, D. Hamilton, "Experiences Using Lightweight Formal Methods for Requirements Modeling," *IEEE Transactions on Software Engineering*, January 1998, pp. 4-14

[5] Wing, Jeannette M. "A Specifier's Introduction to Formal Method"

[6] M. Heimdahl, "Formal Methods for Developing High Assurance Computer Systems: Working Group Report," *Industrial Strength Formal Specification Techniques*, 1998, Proceedings, pp. 60 –64

[7] V. Berzins, "The Design of Software Interfaces in Spec," International Conference on Computer Languages, 1988, Proceedings, 266-270

[8] V. Berzins and Luqi "An Introduction to the Specification Language SPEC," *IEEE Software*, March 1990, pp. 74-84

[9] M. Neil, G. Ostrolenk, M. Tobin, M. Southworth, "Lessons from using Z to Specify a Software Tool," *IEEE Transactions on Software Engineering*, January 1998, pp. 15-23.

[10] P. Alexander "Best of both worlds [formal and semi-formal software engineering]," *IEEE Potentials*, December 1995-Janauary 1996, pp. 29-23

[11] A. Evans, A. Wellings "UML and the Formal Development of Safety-Critical Real-Time Systems," *IEEE Colloquium on Applicable Modelling, Verification and Analysis Techniques for Real-Time Systems*, 1999, pp. 2/1-2/4.

[12] LAMPS MKIII Weapons System Manual, A1-H60BB-NFM-010, 1 May 1997

[13] Federation of American Scientist Website, http://www.fas.org/man/dod-101/sys/ship/weaps/an-slq-32.htm

[14] R.S. Pressman, *Software Engineering: A Practitioner's Approach*, Fifth edition, McGraw-Hill, 2001

[15] Luqi and J. Gougen, "Formal Methods Problems and Promises," *IEEE Software*, January-February 1997, pp. 73-85

[16] J.S. Gansler, The Under Secretary of Defense, Memorandum for Component Acquisition Executives Director of Ballistic Missile Defense Organization, 26 October 1999

[17] Jane's Online Website, http://www.janesonline.com/

[18] J. Herman, M.S. Thesis, "Target Identification Algorithm for the AN/AAS-44V Forward Looking Infrared (FLIR)", Naval Post Graduate School, June 2000

[19] V. Berzins and Luqi, *Software Engineering with Abstractions*, Addison-Wesley, August 1990

[20] I. Sommerville, *Software Engineering*, Sixth Edition, Addison-Wesley, 2001

[21] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*, Prentice Hall, 1998

[22] S. Liu, R. Adams, "Limitations of Formal Methods and An Approach to Improvement," Software Engineering Conference, 1995, Proceedings, pp. 498-507

[23] X. Jia "A Pragmatic Approach to Formalizing Object-Oriented Modeling and Development," Computer Software and Applications Conference, 1997 Proceedings, pp. 240-245

[24] Holloway "Why Enginners Should Consider Formal Methods," Digital Avionics Conference, 1997, pp. 16-22

[25] A. Hall, "Seven Myths of Formal Methods," *IEEE Software*, September 1990, pp. 11-19

[26] M. Shroff, R. France, "Towards a Foramlization of UML Class Structures in Z," Computer Software and Applications Conference, 1997, Proceedings, pp. 646-651

[27] L. Favre, S. Clerici, "Integrating UML and Algebraic Specification Techniques," *Technology of Object-Oriented Languages and Systems*, 1999, 151-162

[28] M. Feather, "Rapid Application of Lightweight Formal Methods for Consistency Analysis," *IEEE Transactions on Software Engineering,* November 1998, pp. 949-959

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, California 93943-5101

3. Chairman, Code CS 1
   Naval Postgraduate School
   Monterey, California 93943-5100

4. Dr. Luqi, CS/Lq 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, California 93943-5100

5. Dr. Man-Tak Shing, CS/Sh 3
   Computer Science Department
   Naval Postgraduate School
   Monterey, California 93943-5100

6. Dr. Neil Rowe, CS/Ro 2
   Computer Science Department
   Naval Postgraduate School
   Monterey, California 93943-5100

7. Matthew A. Lisowski, LT USN 1
   615 Wedge Lane
   Fernley, NV 89408